# read using

No. 17 July 1986

Research Machines
National User Group Journal

**Executive Committe 1986/87**

| | | |
|---|---|---|
| Chairman | Ken Maynard | 7 Wood Drive<br>Stevenage, Herts. SG2 8PA<br>**0438 354809** |
| Secretary | Steve Burrows | Wirral CAE Services Ltd.,<br>Gorsey Lane, Wallasey, Wirral. L44 4HE<br>**051-639-8237** |
| Treasurer | Graham Townsend | Lampton School, Lampton Ave., Hounslow.<br>**01-572-1936**<br>3, Loseberry Road,<br>Claygate, Esher, Surrey.<br>**0372 66043** |
| Membership Secretary | Les Watson | National User Group,<br>c/o Research Machines Ltd.,<br>PO Box 75, Oxford. |
| Editor | Robin Davies | 13 Paget Road,<br>Wolverhampton. WV6 0DS<br>**0902 23735 {evening}**<br>Heath Park High School, Wolverhampton<br>**0902 737328 {daytime}** |
| Librarian | John Ward | The Clarendon Laboratory<br>Oxford OX1 3PU |
| Peter Bishop | | Peter Bishop Associates,<br>23 Larkfield Road,<br>Richmond, Surrey. TW9 PG |
| Tim Davey-Winter | | Computing Teaching Centre,<br>59, George Street, Oxford. |
| John Freeman | | North London Science Centre,<br>62-66 Highbury Grove, London N5 2AD |
| Steve Rochford | | Hounslow Manor School,<br>Prince Regent Road,<br>Hounslow, Middlesex TW3 1NE |
| Keith Ashby | | Ashby Farms, The Buss Barn,<br>Fairfield Road, Appledore, |
| Steve Kennewell | | BECC, The Bordesley Centre,<br>Camp Hill, Statford Road,<br>Birmingham B11 1AR |

**Advertising**

We accept advertisements for **Read Using.** The charges below assume that camera-ready (black & white) artwork is supplied. Any advertisements typeset by **Read Using** are charged at cost. **Read Using** is an A5 format Journal and advertisments are accepted at the Editor's discretion.

| | |
|---|---|
| Back Cover: **£60** | Full Page: **£50** |
| Half Page: **£30** | Quarter Page: **£18** |

# Here it is list


Editorial . . . . . 1
Letters . . . . . 3
Write for Read Using . . . . . 4
Odds and Ends . . . . . 5
Library News . . . . . 7
Review of AlphaCalc and AlphaStat . . . . . 9
Review of Robot Arena . . . . . 15

**FF00 Pages**

Assembly Language Programming . . . . . 20
Using Front Panel to Change filetypes . . . . . 34
Using RUN.COM from any User area . . . . . 39
Patching TXEDE for the Walters WM2000 . . . . . 42
Patching WSMSGS.OVR . . . . . 44

**Programming**

Redefining the 380Z character set . . . . . 47
Programming with WORDSTAR . . . . . 56
Combinations of N things . . . . . 65

**Nimbus**

Feeling at home with your Nimbus . . . . . 70
A Nimbus Case History . . . . . 74
Nimbus Anagrams . . . . . 77

**Communications**

Going on-line . . . . . 80
A simple communications program . . . . . 85

**Advertisers' Index**

Longman Micro Software . . . . . 19
Peter Bishop Associates . . . . . 14
Joel Electronics . . . . . 64
Peter Whittle Associates . . . . . 46,79,84
Software Production Associates . . . . . Back Cover

# Editorial

Well, here it is, the A5 *Read Using*. I hope that you like the new format because it's here to stay! No doubt there are lots of things to get right, but by the time I've produced a couple of issues everything will fall into place! Comments about the Journal are always welcome. I promise to ignore only the uncomplimentary ones. Over the next couple of issues it is intended to develop the scope of the Journal to meet the needs of the Group's changing membership. More material will appear which deals with the ever-growing numbers of Nimbus users. I am especially keen that users of applications packages for the Nimbus, such as Superfile, Word, accounts and graphics packages, will find *Read Using* helping them. To that end, various articles are being commissioned from experienced Nimbus users.

The Nimbus, of course, is not the only computer sold by RML. Although sales of 380Z have virtually ceased as they are basically only made to order, the good old 480Z is still trundling off the production line. Thus it is inevitable that *Read Using* will feature material for the 380Z rather less than in the past. We shall not forget it, however, and I would expect that each issue of *Read Using* will contain material relevant to users of the 380Z for some time to come. I am very interested to hear from members who are using their 380Zs for control purposes. Is *anybody* out there using the VELA?

Thanks to 3 people who wrote offering their services for reviewing material. You may well hear from me in due course! Reviews of software are going to appear more regularly in *Read Using* as software producers see the value of sending me material. Like you, I dislike buying software unseen as it's so easy to buy rubbish. Without pre-empting detailed reviews in the next issue of the Journal, lookout for software published by Cambridge: I've had three packages for evaluation (two Geography and one Home Economics/Biology) and they are good value for money.

**Microlink**

This fairly new facility via Telecom Gold is going to be of interest to us all, as a Closed User Group section for RMNUG members is going to be established. See elsewhere in the Journal under **Comms**.

Can I remind readers that the User Group offers a Help-Line for members via Executive Committee members: telephone numbers at the front! Mind you, we may not know the answer immediately, but we can sure try!

The news about the financial difficulties faced by Apricot Computers should encourage us all to respect RML's cautious approach. However, when AMSTRAD reveal their sub-£500 IMB-PC compatible next month even RML will have to look to its laurels.

Pro-Pascal is up and running on Nimbus at Wolverhampton Teachers' Centre with the graphics extension that should enable the use of sprites within a Pascal program.

Finally, many thanks to all those who sent good wishes following my recent eye operation. For those who like the details, I had a Right Peripheral Iridectomy! The left eye was done 15 months ago.

# Letter to Editor

Arc Computer Services Ltd
Smallfield

Dear Robin

Thank you for arranging for our administration package CORK to be reviewed. May I make just three comments.

Firstly, as your reviewer indicates, the unit of data for Cork is called a 'file'. Once Cork has been run, and a particular file has been loaded, it is not possible to load another file without first exiting from Cork and then reloading Cork. Thus, if your reviewer devoted the whole of one file to a single class, as appears to be the case, then he is correct when he states that to load the data for another class he must first exit from, and then reload, Cork. The underlying problem here is really one of balance. A file can store up to 22K of data. Thus one can have a small number of names (e.g. a class) and a lot of data per name, or a lot of names (up to 250) and less data per name.

Secondly, the documentation. Again, as your reviewer states, comprehensive help notes are accessible from within Cork, and a print-out of these notes is provided. It was originally intended to provide a conventional manual – and we may still do so – but to do so would raise the production costs by about £5 per package (which with a profit element and VAT means about £8 or £9 to the purchaser), and we are of the opinion that at present even such a small saving has become significant to teachers. Striking a balance between adequate documentation (your reviewer described ours as 'copious'), and documentation which is a work of art in itself, is a problem faced by all software producers, and we certainly would be pleased to learn of other peoples' opinions.

Finally, the price. Because of my inability to do simple sums I told you a price of £48.50. In fact the price is £48.30 which equals a basic price of £42 plus VAT of £6.30. My apologies.

Yours sincerely

Brian Phillips

# Writing for Read Using

Please do! We welcome almost anything that might be of help or of interest to users of RML equipment. Especially useful are those articles, letters, reviews about your discoveries concerning the Nimbus. All material will be considered but naturally enough, not everything is published.

Don't worry about your level of expertise: we need people writing about their problems, mistakes, worries and successes at all levels. You may have written a useful little routine in PASCAL or SBAS: share it! Perhaps you've cracked the problem of interfacing the Super-Whizzo printer to the Network server: send the details. General articles about the use of computers in any school situation will be of interest, especially in primary schools. If you've bought a program that's turned out to be a success, send in a review. We love graphics output from your printers/plotters.

If at all possible, *send material on disk*. This makes life easiest and disks are returned with your cheque! Don't worry about formatting. Plain old hand written material is welcome. Printer output that would aptly illustrate your article/idea is most welcome. Try to use a new or good ribbon and indicate clearly where in your article any illustrations would be best placed. There's now no point in sending camera-ready articles as the Journal is now typeset from disk.

We try to pay £5 per full page of text and *typically* £2.50 per printed page of program listings, output and the like. The Editor's decision on all remuneration and publication matters is totally undemocratic! (or final).

Copy deadline for *Read Using 18* is September 12th and the next issue should be published early in October.

Copy deadline for *Read Using 19* is November 7th and that issue should be published by mid-December.

Copy deadline for *Read Using 20* is January 31st, 1987 and that issue should be published early in March – just before the AGM.

# Odds and Ends

The Spring *Nimbus Educational Catalogue* – an item of over 100 pages – shows how far educational computing has travelled in the last couple of years. Far from being just a collection of little programs for the kiddies, it shows how seriously computing in educational establishments is being taken, both by software and hardware suppliers. It also is an indicator of how programs are rapidly appearing for the Nimbus. A copy should have been sent to your school, but if you ring RML they will send one.

With communications by computer gathering pace we have heard from *The Electronic Tourist Board*. You haven't! Well, it was new to your Editor as well, but you can find it by dialling 0874 730172. The ETB emulates Prestel and requires a modem running at 1200/75 baud. They also have a Prestel mailbox: 874730692. No charges are made to callers, except the normal telephone connect charges. What does the ETB offer? Well, sources of information about South, West and Mid Wales. Information providers include British Rail, Gwent and Dyfed County Councils and myriad smaller organisations such as hotels, camping parks, etc.

*Microft Technology*, of ASPECT fame, write to tell us about Clam, their new program to protect sensitive data from unauthorised acces. It requires a Nimbus, *Menugen* and MSDOS 2.0. Existing *Menugen* users can upgrade for £60. .

The upgrade for 8-bit, 480Z TCL Pascal is at last available. The upgrade is free and those who wrote to RML about it will hear from them direct. If you want a free upgrade, send your original master disk to Technical Support with a covering letter. You are asking for TCL Pascal V4 for 480Z networks. This upgrade is *not* available for 380Z.

By now, Technical Support ought to be available on a direct line: Oxford (0865) 728384 – but don't shout at me if that isn't yet working!

Users of Chain 64 Networks should check with Technical support about the usage of versions of MODWINI. In fact, all users of Chain 64 ought to ask for a copy of *Technical Support Bulletin 7* as it contains important information about using Chain 64.

Users wishing to increase the speed of disk access under MSDOS should alter the CONFIG.SYS file to increase the number of buffers to 16 for floppy based sustems and 32 for winchesters. Note that this represents 8 and 16K of memory and may represent a problem for 192K systems.

Our friends at SPA have released some more goodies, including a thoroughly uneducational and therefore highly sought-after game called *Barrels* which runs on 480Z and Nimbus. Also for 8-bit machines is *View*, a local Viewdata system and CPU-SIM, a central processing simulator. Also released are two extensions to RM Nimbus BASIC and one for Nimbus LOGO.

The User Group Library is working in the ways and means of providing MSDOS public domain software and details should be available in the next edition of *Read Using*.

# Library News

## *John Ward – generic Librarian*

£1.50 per volume + £1.50 unless you supply a single density formatted disk.

(+ £2.50 for 8" or Nimbus (!q.v.) disks) Send your request with your membership number and a cheque payable to the User Group to the librarian – address inside front cover.

There is a lot of good software in our library. There may already be a program to do what you want but you will never know unless you send for volume 0. This volume contains lists of the entire contents of our library and a utility called FIND to help you search. You can print out the lists but if you know at least part of a program's name or description FIND will find it quicker.

David Jackson in Read Using 16 wanted to know how to get one COM file to load another in order to do some menu writing. I expect someone will oblige but he really would be much better off with ZCPR2 under which menus are practically built-in. The Editor has recently sent for ZCPR2 on volumes 15,16 and 17 some we might see some comments from him. (*You'll be lucky!*)

Talking of the Editor, he must be getting really short of printable contributions if the club's chairman has to write articles explaining how he has mis-understood part of my ZCPR2 description in Read Using 13. GO might be just an 'empty' program to the chair but in ZCPR2 it is a built-in command just like DIR or ERA. Therefore it works much faster and does not take up directory space. To the bottom of the class Maynard and *go* and read what I wrote again! (*15 all*)

Peter Arnold had a useful utility called USERDIR in Read Using 16. Before brave souls set to and type in all those pages of machine code they may like to try SD.COM on volume 14. I think SD will do the same thing and sort the output alphabetically into columns. Good to see Peter writing generic CP/M code and not using EMTs though. And he put a nice title at the front so it is easy to see what the program is called and what revision it is. A very good habit some of the other machine-coders would be wise to acquire.

Some of you must read what I write because I have received quite a lot of software for the library recently. Mr B.C. Pierce of Queen Elizabeth School, Corby has contributed 3 disks worth! So by the time you read this there should be at least 3 new volumes in the library. The catalogues will be in the next Read Using, (because I haven't done them yet) but the programs will include word games (anagrams, spelling and flashcard), a printer for Multiplan, a Cesil interpreter, a game called Nightmare Park and an encoder/decoder program to 'deter hackers and keep data confidential'.

Dr Cecil G. Armstrong of Queen's University, Belfast has sent me two volumes worth of Nimbus programs. There is a Turbo Pascal graphics library which has been adapted from the Pro Pascal graphics library and another library which gives 'source code compatibility with the IBM PC version of Turbo for screen handling, windowing graphics etc.'. Another set of programs allow different fonts to be defined and used and printed with word processors such as Word. Dr Pierce would like this distributed as 'shareware' i.e. if you like it please send a small contribution. This is the first Nimbus software I have received. I would like to think a bottle of bubbly is on its way to you Cecil but unfortunately it is not. Also, now, Centec have contributed a memory test program for add on RAM cards (RM10). With this unexpected plethora (I bet that is not in my spelling checker) of new software there had to be a snag. Guess who does not have a Nimbus? N.B. RML. Also I don't have MSDOS versions of CRCK the checksum program, or the squeezer and unsqueezer programs. Nevertheless, enthusiasts may send for pre-release versions but send £2.50 if you don't send a disk. Nimbus owners get a much better deal from the library as one volume contains about 5 times as much as a 4/380Z volume. Perhaps we should raise the price?

I have been writing an information system for our annual Open Days. It is a menu system where the user just presses one key to find out about laser tours, the latest in crystalographic techniques or whatever. You've guessed it – it was really simple under ZCPR. All I had to do was write each text page and the rest almost wrote itself. At least 200 380Z owners out there just don't know what they are missing. Don't be a closet 380Z owner (or keep your 380Z in one) join the CP/M renaissance with ZCPR2 on volumes 15, 16 and 17. Of course if you have abandoned your RML machine for a BBC you will definitely need a copy of the BBC to 380Z transfer utilities. Come on now it's your money I'm after. You don't think I get paid for writing this do you?

# AlphaCalc and AlphaStat
## from Peter Bishop Associates
### *A Review by Roger Blackford*

These two packages are the first two PBA software products released for the RML range of computers.

Each package is accompanied by a User Guide and a set of worksheets whose intention is to give specific instructions for the use of AlphaStat and AlphaCalc to carry out a particular task.

In using these packages I was impressed by the amount of work that had gone into their preparation and by some of the excellent ideas featured in the worksheets. However, I feel that there are some serious weaknesses in the packages – especially from a user's point of view and I will outline them below.

**AlphaCalc**

The preliminary instructions for creating a 380Z/480Z master disk and installing the package are clear and concise. Instructions for running the package appear to be incorrect. The *Transfer* command is not available at the start and the user is always given the last spreadsheet used. This is very annoying and most certainly is not the way that spreadsheets are actually used. The idea of keeping a file of spreadsheets is that they can be continually updated as new data is received. It also means that if the package is used with several classes, each class has the spreadsheet of the previous class loaded first. The pupils then have to load the spreadsheet that they require.

The format in a formula is inconsistent. The first specimen worksheet indicates the formula:

```
R 4C 6 = R4C1*R4C3
```

Note the spaces before the row and column numbers on the left but not on the right.

I found the numbering of the columns poor and their alignment to the right instead of being centred unhelpful. Also a weakness was, I felt, the use of the arrow keys only to select the choice. The ability to type a letter to select the action would make the spreadsheet easier to drive.

Having used Multiplan a great deal on the 480Z for a while I found it tiresome to have to enter data by typing <RETURN> *and then* have to move to a new cell. It is better, I feel, to be able to enter data by moving the brackets to another cell. I noticed that when the brackets are in the first column, the left bracket is missing.

I found that AlphaCalc is very slow in transferring spreadsheets to and from disk. A really useful facility is the ability to scroll part of the screen, leaving labels and headings in place. It's easy to do this. You can load a whole spreadsheet or just its template, which is extremely good.

If the print option is chosen, the program prints the first 8 columns by 32 rows as page 1. Columns 9 to 16 are only printed if they have been used and are then labelled as Page 2.

The package allows the user to enter numbers, labels and formulae into the cells. Columns and rows may be inserted or deleted easily using the Edit command. Formulae are adjusted when this is done.

16 spreadsheet ideas are provided with the package, although not all were on the version I used. These ideas are really very good but I feel that they haven't been presented very well. Generally the screen layout is bad in most of the spreadsheets provided with far too much on the screen at once. Far more consideration needs to be given to the way in which pupils react to methods of presentation and to the amount of information they are capable of absorbing.

### ACWS 1 Frequency Distribution

The screen display is untidy and would be confusing to the average 13 year old for whom the spreadsheet was designed.

### ACWS 2 Mystery Numbers

The process of printing the spreadsheet detracts from this application as it takes too long to print several different examples. This is essential if the pupil is to ensure that the right formula is predicted. The third activity, which involves entering formulae in 7 boxes, is very tedious and prone to error.

### ACWS 3 Parkway Station

An excellent idea but not for all 13 year olds. The more able 14 year olds would find this problem interesting and worthwhile. The more academic parts of this investigation are more appropriate to more able 15 year olds.

### ACWS 4 Football League

Another example of a good idea spoilt by poor layout. Why not keep the details of the number of points for a win etc., away from the League Table?

### ACWS 5 Fibonacci Sequence

A good idea which has again been rather spoilt by poor screen layout. Placing the ratio of two successive terms below and to the *left* of the terms is very confusing. Children younger than 15 (suggested age) could use an improved version of this worksheet to advantage.

### ACWS 6 Tuckshop

This was not on the disk provided.

### ACWS 7 Cuboids

A mediocre idea.

### ACWS 8 Approximate Solutions of Equations

Uses Newton's Method to solve equations iteratively. Can be used to show convergence to a solution quite well. For 6th formers the spreadsheet is easy to use.

### ACWS 9 Hire Purchase

Not supplied.

### ACWS 10 Magic Squares

Provides useful practice in addition for 11 year olds of all abilities. The screen layout could be improved.

### ACWS 11 Differentiation

A very useful spreadsheet. Easy to use and change.

### ACWS 12 Pocket Money

A very good idea spoilt, in my opinion, by putting too much on the screen at the start. The average 11 year old will be confused by the amount of data first presented. If the spreadsheet had started with fewer columns, and the pupils had been allowed to develop it, the educational value of this example would have been far greater.

### ACWS 13 Invoices

A very useful activity for 13 year olds, but spoilt a little by a very cluttered screen. This is surprising as there is plenty of room to allow spaces between some of the columns without having information off the screen. Three of the columns are wasted.

### ACWS 14 Bank Accounts

A useful activity for 13 year olds, but again a cluttered display.

**ACWS 15 Population Growth**

A very useful spreadsheet to teach population growth. Unfortunately, it gives too much information at the start. It would have been better if the pupils were allowed to develop the spreadsheet themselves after the initial data was displayed. It doesn't give pupils the chance to discuss the effects of rates of growth on the population from the start. The book doesn't suggest extending the use of this type of spreadsheet to rates of decay.

**ACWS 16 Population Dynamics**

A very useful worksheet to enable pupils to model the growth of a population. More able 16 year olds could find it quite stimulating.

The documentation provided for the worksheets is well produced and generally easy to follow.

**AlphaStat**

Installation instructions are, like those for **AlphaCalc**, clear and easy to follow.

I found the instructions for using the program very unclear and found out how to use the program by trial and error.

The program allows the user to create tables of data and then perform calculations on the data. The tables consist of 8 columns of 64 rows plus a column for labels. The software is menu driven in the same way as **AlphaCalc** and has the same drawbacks. The choice from a menu cannot be done by typing a single letter and the screen layout in the specimen worksheets is far too crowded.

The use of the Edit command allows the user to insert and delete labels, rows and columns as well as allowing the insertion of data. Files of data may be read from and written to disk and the user can print tables of data.

The Arithmetic option lets the user use variables and formulae. The Statistics option provides facilities for calculating means, correlation coefficients and frequency counts, as well as a sort facility. The user can display data as line and bar charts and can download data from the Presetl educational database.

There are many other pieces of software on the market which provide similar facilities and are easier to use.

As with **AlphaCalc** the big advantage of this package is the number of worksheets provided, all of which provide good ideas for the teacher. But I feel that they are spoilt to some extent by poor screen layout and by trying to display too much data on the screen at once.

**ASWS 1 Population Densities**

Very difficult data for 13 year olds to use

**ASWS 2 British Climate**

Very poor screen layout. This worksheet illustrates the difficulty of choosing the data to be plotted on the x- and y-axes.

**ASWS 3 Pulse Rates**

Good idea that needs a less crowded display.

**ASWS 4 Looking at Functions**

A very good idea which is a little spoilt with too many x-values at the start. The plotting of the graphs of the functions is clear and very useful for investigating functions.

**ASWS 5 Period of a Pendulum**

A good idea, but the screen is rather crowded.

**ASWS 6 Gas Consumption**

This worksheet is supposed to be for 14 year olds of all abilities but presents the pupil with a very crowded screen of sophisticated data.

**ASWS 7 Weather Records**

Again, I feel that far too much data is presented far too soon.

**ASWS 8 Household Expenditure**

A good idea, but with too much data presented at the start.

The following worksheets provide the teacher with good ideas but the implementation of the ideas is somewhat spoilt with crowded screens.

**ASWS 9 Motion Under Gravity**
**ASWS 10 Capacitor Discharge**
**ASWS 11 Semiconductors**
**ASWS 12 Boyle's Law**

The packages are available for all RML machines, stand-alone or networked. The Nimbus version requires at least 256K. The packages are sold either under LEA licence or single user licence. The cost for an LEA licence is naturally related to the number of institutions. The cost for single users is:

**AlphaCalc**: £39.00
**AlphaStat**: £49.00

Peter Bishop Associates say that the software is fully supported and queries from users are welcome.

If what I feel are serious weaknesses in usage, screen presentation and instructions can be dealt with, then these two packages are likely to be of considerable value. As they stand, the ideas and worksheets provided are of great value and those provided for **AlphaCalc** show clearly how a spreadsheet can be more that just an accountant's friend.

## AlphaStat — £ 49.00 + VAT

*A powerful mathematical, statistical and graphics software pack which operates on tables of numbers – 8 columns, 4 rows:*

**Mathematics:** Formulae can be evaluated over entire columns.
**Statistics:** Mean, median, mode, standard deviation, correlation, rank correlation, sorting and frequency counts.
**Graphs:** Line or bar graphs of numbers in any two columns.

**Subjects:** Physics, Chemistry, Biology, Geography, Economics, Sociology, Mathematics.
**Computers:** RML 380Z, 480Z and Nimbus. The same versions run stand-alone and on networks.
**Availability:** Now

---

## AlphaCalc — £ 39.00 + VAT

*A versatile spreadsheet which combines ease of use with the power of a full commercial spreadsheet:*

On-screen entry and editing of numbers, labels and formulae.
Full formulae evaluation using row and column notation.
Windowing facility to fix part of the spreadsheet on screen.

**Subjects:** Business Studies, Mathematics, Computer Studies.
**Computers:** RML 380Z, 480Z and Nimbus. The same versions run stand-alone and on networks.
**Availability:** Now

---

## DeltaStat — £125.00 + VAT

*A very powerful mathematical, statistical and graphical software pack for the analysis of research data. DeltaStat uses tables of 8 columns and 1024 rows and includes the following facilities:*

**Mathematics:** Formulae can be evaluated over entire columns.
**Statistics:** Weighted mean, median, mode, standard deviation, correlation, rank correlation, sorting, frequency counts and t-test.
**Advanced Statistics:** Linear curve fit, statistical curve fit (binomial, Poisson, exponential and normal distribution), chi-squared test, analysis of variance.
**Fourier Transform:** Discrete Fourier transform of up to 1024 data points.
**Graphs:** Line and bar graphs of one or two variables plotted against another variable.

**Subjects:** Research in natural, social or medical sciences. Administrative uses such as tabulation of examination results.
**Computers:** RM Nimbus. The same version runs stand-alone and on networks.
**Availability:** 21st July 1986

**Peter Bishop Associates**
**23 Larkfield Road**
**Richmond**
**Surrey** **TW9 2PG**
**01 948 4382**

PBA

# Robot Arena from SPA

*Review by Fintan Culwin*

Perhaps I'm getting old and jaded, perhaps I'm just getting old. But I see fewer and fewer educational programs which cause me to believe that we are actually using the technology for the best advantage of the students with whom we work. I could probably count less than a dozen programs in ten years which have caused me to stop in amazement and wonder at the educational aims which can be met.

The archetype of this reaction must have been my first encounter with LOGO and shortly after, the concept of a microworld. It became possible to consider programming as an educational activity, rather than the mid-century equivalent of irregular Latin verbs. The only other package that has come even near LOGO within my experience is DEVELOPING TRAY. The package reviewed here came very close to landing in my estimation between TRAY and LOGO. However, in my opinion it is almost totally unusable in the classroom.

ROBOT ARENA is an attempt at a microworld, one of the most convincing I have seen. I would like to believe it had been written in LOGO but I suspect it was written in assembler using SPA's superb high resolution graphics routines.

The scenario for the microworld is an arena consisting of 80% of the 480Z HRG screen. Within this arena up to four robots can be employed as gladiators, firing missiles at each other until only one remains; until it too runs out of energy and expires. The robots are controlled by programs written in a programming languague called ROBOL.

ROBOL is structurally very similar to LOGO. Robots can be commanded to move (forwards, backwards & rotate). They can investigate the arena ( Radar ). They can affect the arena (launch [a missile], laser [to defend against a missile], shield [to defend against anything]). System variables are available to inform the program about the state of the arena (Xpos, Ypos, heading, armed, energy, object, others). ROBOL contains facilites for loops (repeat/again) and for making decisions ( if/then/else/endif). ROBOL primitives can be built into user named procedures taking arguments (to(). . . .return). An improvement on 480Z LOGO is the comment facility, even a limited degree of recursion is possible. For a simple idea of the programing capabilities of ROBOL consider a LOGO with multiple turtles, and specific words for arena activities pre-defined.

The programming environment is equally important for educational useage. Here the design is a little like TCL PASCAL. A simple menu screen gives access to commands to save or load ROBOL programs. To use a rather sophisticated editor to write, change or print ROBOL programs. To enter the arena with one or more robots, and even a single step debugger to trace programs. The only irks I found were no facility to transfer a ROBOL program from one robot to another, without having to save the program as a file first. Also no facility is provided to display a directory of ROBOL programs from the disc.

Network users will find that files are closed correctly allowing ROBOL programs to be exchanged between stations; and the creations of the programs do battle with each other.

The package contains two manuals, a teachers guide and a pupil's workbook. The workbook is up to SPA's usual high standards giving a friendly introduction to the arena, and programming in ROBOL. The teacher's guide is a comparative disappointment being short on educational content, comparatively terse and missing such essential adjuncts as a glossary of reserved words.

The arena display itself is technically excellent. The robots move smoothly and quickly. Each having an updated display showing the status of the robot. In debug mode the four line scroller shows each command of the program as it is executed, but not allowing such things as breakpoints or displaying the value of variables as they are used.

From a purely technical point of view this is a superb package; making the best possible use of the 480Z's HRG. With a well designed and useful user interface, supported by well written, well thought out documentation. It is a great pity that I will not use it and cannot recommend that it be used.

*What follows is a personal, philosophical viewpoint that is not connected with the technical aspects of the package.* **Editor**

The technology that we use is effectively neutral. The same hardware and software designs that we use can be employed to control the passage of missiles on to targets or control the passage of current into a heart pacemaker. It is the context that the systems are used within which determine if they are used for destructive or constructive purposes. The context of this system is essentially destructive. It compares badly with games such as space invaders. In the context of an action 'shoot them up' game the user's action is essentially passive and consists essentially of fast learned reactions. Within the context of this game the user's actions consists of powerful procedural problem solving thinking and learning being applied to actively destroy the robot programmed by a fellow student.

The measure of success within the arena is to destroy as many other student's products as possible, while sustaining as little damage oneself. This is not the ethos I want to encourage within my computer rooms. When my students are exploring LOGO, communication and knowledge is open and constructive. Ideas and products are displayed with creative pride and shared between groups. Such an approach within the arena would be 'defeatist'. The ethos would be one of competition and knowledge once discovered would be hidden and shielded from other students; displayed cryptically only when the robot enters the arena. This is the twentieth century electronic equivalent of a Roman gladiators arena. I would rather teach irregular Latin verbs than encourage the ethics of the arena within my students.

There is another consequence of the ethos of the arena entering the computer rooms. Although I do not know of any actual evidence, I strongly suspect that sex differences between female and male student's perceptions and use of computers exist. Female students may perceive the school's computers as being tainted with the same views that are held concerning maths and physics. The usage of computer programs by male and female students in the open sessions that I run are noticeable. Female students are more likely to use an expressive or exploratory program. Male students are more likely to use defined or action programs. ROBOT ARENA has the capacity within it to be a very expressive environment but the context of agressive competition is likely to alienate the female students and dominate the males. Exactly the models of behaviour which underpin the differences that result in female students turning off from technology, including computers within our schools.

The raw potential of the idea is enormous. In addition to the procedural thinking employed as a consequence of the LOGO type languague. The arena is essentially a public place where the student's programs can be displayed alongside each other and the effectiveness of the ideas which produced them measured. It adds a structural social dimension to an activity which too often is presented and experienced as being individual and intellectual.

The effort which SPA put into producing ARENA could equally effectively have been put into a less agressive action type environment. For example the robots could easily have become robot scout ships competitively or cooperatively hunting for salvage within destructive asteroids. Or it could have been expressed context free giving us the first mutiple turtle LOGO for the 480Z.

Technical expertise is not enough the uses that the products are likely to be put to must also be taken into account. The consequences of using this package will be a further identification of computing and masculinity. While the overwhelming majority of computer studies teachers are male packages like this will continue to appear and no doubt be used. While packages like this are being used it is less likely that our female students and colleagues will become seriously interested in computers.

*More from the Editor*

Whilst I can see the points that Fintan is making, I believe that most of them are rubbish (I said that to him at the AGM!). I could, as the father of 3 teenage daughters (with one in HE taking a Computer Studies course ) and a son make several observations about girls, Maths & Science. However perhaps our readers would like to respond. I feel that ROBOT ARENA is an excellent package and in the next issue of *Read Using* I hope that we will be able to publish a further review by teachers who have actually used the package in the classroom. Meanwhile, write a letter if you have a view. Note that I don't intend *Read Using* to become a forum for politicising computing!

Finally, whilst I think that much of what Fintan says above is rubbish, I love his software!

Robin Davies

Longman Micro Software

**Now available from Longman Micro Software RML 480Z network versions.**

We have produced RML 480Z network versions for some of our newest and most exciting software titles.

*Here are some of those titles.*

## Understanding Economics Pack 1

This new pack contains Workers and Machines, Estate Agent, Iron and Steel Location, Balance of Payments and Break Even.

*- (all these titles are also available separately).*

Pack price for 480Z net £50.00 + VAT
Individual titles £27.50 + VAT

## Microdiet

Microdiet is the ideal tool for the classroom study of nutrition. The program contains a database of over 500 foods with details of up to 29 nutrients for each food. There are various ways of interrogating the database and it is fully supported by comprehensive documentation.
£37.50 + VAT - RML 480Z net

## Dynamic Modelling System

A very powerful general purpose modelling system which enables the user to investigate his or her own models or one supplied as part of the package. The system provides graphical or tabular output and shows the relationship between any two variables in the model.
RML 480Z network version £45.00 + VAT

*We have network versions available for many of our software titles including geography, history as well as science related subjects.*

If you're interested in further details on any of our titles published for the RML network machines then please contact Denise Pearce, Micro Software Unit, Longman House, Burnt Mill, Harlow, Essex CM20 2JE. Tel: Harlow 26721 X3371 and quote ref RU/RML.

# Assembly Language Programming

*by Andy Brimble*

Paddington College, London

Assembly language programming may seem at first to be a daunting prospect, but it is very rewarding if you can stick with it, and is not nearly so difficult as you might imagine (after all, if I can do it, anyone can !)

I am setting out here to try and de-mystify some of the concepts surrounding assembly language programming, and to give you a gentle nudge into the field, so that you can tackle your own projects with more confidence. Obviously in an article such as this, I can't possibly get you to become an advanced assembly language programmer, but I think you will be able to achieve this if you can just take the initial step.

The assumption is that you have access to an RML 380Z or 480Z, with the following software :

1. An editor such as TXED or WORDSTAR
   This will be used to produce programs in the form of files of text. You must be confident enough to use the editor of your choice (I find WORDSTAR best – sorry RML!) to enter, save and edit files of text. If you use WORDSTAR you should choose the NON-DOCUMENT mode to edit your files. (You can type programs in any language using WORDSTAR like this – try typing a few BASIC programs – they should run just as well as those you produce using BASIC's line editor)

2. ZASM.COM and the ZASM manual

This is RML's own Assembler, which will take your programs and convert them into programs which will run on their own.

You should be fairly familiar with CP/M and its commands such as DIR, ERA, REN, TYPE etc. and the naming of files and drives, how to use files on other drives and so on. It will help if you are familiar with BASIC or another programming language. You should be conversant with the hexadecimal number system (don't panic – there is a bit of revision material on this later !) and the ASCII code system.

It will also help you later to have a book on Z80 programming such as one of the following :

| | |
|---|---|
| *Programming the* Z80<br>(expensive but quite comprehensive) | by Rodnay Zaks (Sybex books) |
| *Z80 Machine Code for Humans*<br>(cheap but a little mystifying) | by Toothill & Barrow (Granada) |
| *Teach yourself Z80 Assembler*<br>(cheap and very communicative!) | by Paul Overaa (Century) |

There are some very good RML publications which will also help you :

| | |
|---|---|
| 380Z/480Z Machine Language Programmer's Guide | PN 11068 |
| ZASM Manual | PN 11066 |
| 380Z/480Z Firmware Reference Manual | PN 10971 |
| CP/M and CP/NET programmer's Guide | PN 12084 |

**Machine code and Assembly Language**

The terms Machine code and Machine Language are often confused with Assembly language, but the two are very different. Machine code is the actual numerical code for instructions which the processor will perform. For example, if we take the instruction which the Z80 uses to ADD a number to a number it is already holding, the Machine code instruction is 11000110 in the binary terms that the microprocessor really works in, but we are lucky enough to be able to use the hexadecimal form of this number if we wish to program in machine code, so the code becomes C6 (hex). If we wished, we could enter individual codes in hex, and data in hex through the front panel of the computer (which is just another editor), and in fact this is quite well explained in the manuals mentioned above. After a few hundred instructions however, the feeble human mind becomes boggled with hex numbers and addresses, and yearns for words and names. Also, if you have just typed 200 hex codes into the machine, and you make a mistake, it is very difficult to find out where it is, let alone rectify it.

Assembly language is the solution to our problems ! It is an agreed set of words and symbols (laughingly called MNEMONICS) which allow us to enter programs in terms which are more human than hex codes. Each mnemonic corresponds with one machine code instruction, and in fact the instruction referred to above has the mnemonic ADD – much easier to understand and remember than C6 or 11000110 don't you think ? An ASSEMBLER such as ZASM is just a program which takes each mnemonic and converts it into a hex instruction. It also handles data in a flexible way, (for example, allowing the use of good old decimal numbers ) and allows us certain other facilities which make it much easier to write programs which are understandable and maintainable.

Unfortunately, the mnemonics are still only easier to understand versions of the machine code instructions, so we must still know what instructions there are and how they work before attempting to write programs with them. This is why high level languages were introduced. They make it possible to write programs in a sort of stilted English which can be translated by special programs (called compilers and interpreters) into the machine code instructions which the computer uses.

So in order to understand assembly language programming, an understanding of the arrangements within the microprocessor is required. Some of the terms you will find in this description of the Z80 are likely to apply to other microprocessors to a certain extent, since most microprocessors work according to well established principles laid down by Von Neumann and referred to as the Von Neumann architecture. This involves the concept of a stored program, consisting of a list of instructions, and a set of data on which the instructions will operate. The instructions and the data are stored *together* in the memory of the computer, which can lead to quite a confusing situation if you are not pre-warned !

A microprocessor is a device that has a set of internal REGISTERS some of which are reserved for special functions, and a set of INSTRUCTIONS which will be carried out on data. Each microprocessor has its own set of instructions so unfortunately, learning Z80 Assembly language won't enable you directly to program other machines such as the 6502 (which is found in the **C Micro!) The DATA can be held within the processor, or in areas of memory (RAM – Random Access or Read/Write Memory or ROM – Read Only Memory) outside it. The memory is organised into a collection of 8-bit MEMORY LOCATIONS, each of which has a specified unique ADDRESS. Thus each memory location can hold one 8-bit value, which could be an encoded character, an instruction, or a numerical value. The address for each location consists of a 16-bit number. This gives a total of 65535 possible addresses, ranging from 0000(HEX) to FFFF(HEX) or 64 kilo-bytes of memory addressable by the processor.

The locations could contain any hexadecimal number between 00 and FF. This number can represent any of the following things :

1. An alphanumeric character (via its ASCII code number)
2. A control character (ditto)
3. An eight bit number (between 0 and 255)
4. A Z80 instruction code

Exactly what it represents depends on the context !!!!!

Here is a block of memory locations with its contents:

```
ADDRESS    CONTENTS
 (in    Hexadecimal)
 _____________
| 0100 |    C9     | <--- a Z80 instruction (return)
|------|-----------|
| 0101 |    50     | <--- ASCII code for 'P' (could also be the
|------|-----------|                          number 80 decimal)
| 0102 |    72     | <--- ASCII code for 'r'
|------|-----------|
| 0103 |    65     | <--- ASCII code for 'e'
|------|-----------|
| 0104 |    73     | <--- ASCII code for 's'
|------|-----------|
| 0105 |    73     | <--- ASCII code for 's'
|------|-----------|
| 0106 |    1F     | <--- control character (31) clear screen
|______|___________|                          on RML machines
```

Here are some examples of memory addresses and their contents, taken from a region of memory mapped using a program which scans each address in RAM, then prints the address and the HEX and / or ASCII contents of that address : the figure given as the location is the hex address of the *first* location on that row, the other locations have not had their addresses printed. Thus the address of the location which contains 48H (ASCII H) in line 1 is 3f83.

```
LOCATION :          CONTENTS:

3f80 |   de   f1   18   48   54   68   69   73<-Hex contents
         .    .    .    H    T    h    i    s <-ASCII characters
3f88 |   20   76   65   72   73   69   6f   6e
                                             <-Fill in
3f90 |   20   6e   65   65   64   73   20   61
                                               <-Fill in
3f98 |    _    _    _    _    _    _    _    _<-Fill in
          n         R    M    L         4    8
3fa0 |   30   5a   0d   0a   ff   0c   1b   3c
          0    Z    .    .    .    .    .    =
```

Note that the first line and the last two have had their contents printed, with an ASCII character if the value stored at that location has an ASCII equivalent, or a "." where there is no ASCII equivalent.

Try to work out the contents of the locations between 3f80 and 3f97, and put them into the diagram.

Try and find the address of every location which contains the value 20H. What ASCII character does this represent ?

Try and fill in the missing hex values in locations 3f98-3f9f.

You will have realised by now that you will need to understand the hexadecimal number system and the ASCII code system if you are going to get anywhere with assembly language. There are a number of books which will help here, and I would strongly recommend that you try to master these before proceeding any further.

## Registers

These are places within the processor chip that will hold binary information. With 8 bit processors, the word length is said to be 8 bits, or one BYTE. The registers in the Z80 are therefore capable of holding one byte of information. The instruction mentioned above , (11000110) takes up one byte.

All processors have a special register called an ACCUMULATOR which is used for arithmetic and logic operations such as addition, subtraction and comparing two values. You can't divide or multiply directly, but have to write subroutines to do this. This is a common feature of microprocessors.

All processors have a **Program Counter** register (**PC**) which tells the processor from which location the next instruction will be fetched. Most processors will also have an area of memory used as a **Stack** for temporary storage of values. There is usually a **Stack pointer** register which keeps the address of the top of stack. Often the contents of all registers are **PUSH**ed onto the stack during an operation, and **POP**ped off it afterwards. This means that operations which affect the contents of these registers can be done without disrupting other operations in the main flow of the program.

A selection of bits in the processor are affected by operations that have been performed, and these are called **Flags**. They are grouped together in the Flag register, and are often referred to as the **Program Status Word**. For example, if an operation resulted in the accumulator containing zero, the zero flag would be set to 1 (indicating that a zero *is* present – the convention is that 1 represents true, while zero represents false).

There is also the facility to combine two eight bit registers to store addresses, or other 16-bit values.

The Z80 is a typical 8-bit processor, with two sets of 8-bit registers, (one set labelled the alternate set) and some special 16-bit registers used for the stack pointer, program counter and the index registers.

The registers are labelled as follows :

### Working 8-bit registers

| | |
|---|---|
| A and F | Accumulator and flag registers |
| A'and F' | Alternate set |
| B,C,D,E,H,L | General purpose registers |
| B',C',D',E',H',L' | Alternate set |

## Special registers

| | |
|---|---|
| SP | Stack pointer (16 bit) |
| PC | Program counter (16 bit) |
| IX | Index register (16 bit) |
| IY | Index register (16 bit) |
| I and R | Interrupt vector and Memory refresh registers |
| IFF1 and IFF2 | 1 bit flip-flops |

In assembly language, we can refer to the registers by these letters, and ask for data to be put into them, or taken out, or even modified while in place. There is a set of instructions for example which LOAD bytes into registers, the mnemonic here is LD.

It is easy to use two registers together by referring to them as BC,DE,HL and even (occasionally) A and F are used together as AF. The most commonly used register is the accumulator A which is perhaps the most useful, since it is possible to perform additions and subtractions within it. A typical operation, which we will look at in the example program, is to load a value into A. This is done very easily by using the instruction LD A, 65 (for example). In this case we could be using 65 to mean the Alphanumeric character A, or we could be interested in it purely as a number. (Don't worry, you'll get used to it !)

The following is a map of the registers in a Z80.

### 1.8-Bit Register Set

| FLAG REGISTER (F) S.Z.*.H.*.P/V.N.C | ACCUMULATOR |
|---|---|
| B | C |
| D | E |
| H | L |

### 2.Alternative 8-Bit Register Set

| ALTERNATE FLAG SET (F') | ACCUMULATOR' |
|---|---|
| B' | C' |
| D' | E' |
| H' | L' |

## 3. Special Registers Including 16-Bit Registers

| | |
|---|---|
| INTERRUPT VECTOR (I) | MEMORY REFRESH (R) |
| IX | 16-BIT INDEX REGISTER |
| IY | 16-BIT INDEX REGISTER |
| SP | 16-BIT STACK POINTER |
| PC | 16-BIT PROGRAM COUNTER |

The Interrupt vector (I) is used by the hardware to control interrupts, which are breaks in the normal sequence of instructions in a program. This is done for a variety of purposes, usually concerned with input/ output to other devices. The 1 bit flip flops are used in conjunction with this register to control input/output. The memory refresh register (R) is used to maintain areas of dynamic RAM intact. These two registers are not usually used by programmers, except to test for error conditions in hardware.

### Flags

These are very useful, and can be very confusing when first encountered ! Assembly language programmers make use of flags to do all sorts of devious things, usually associated with conditional jumps (IF . . . THEN in BASIC). These are the Z80 flags :

S — The sign flag is set to zero if the sign of the number in A is positive. (If the most significant bit of a binary number is 1, then that number is negative, if zero it is positive – thus only 7 bits in an 8-bit number, and 15 bits in a 16-bit number are used for the magnitude, the msb is used for the sign.)
The number 25 in this form is 0 0011001
while -25 is 1 0011001

Z — The zero flag is set to 1 (careful !!) if the contents of A is zero – a useful test for conditional jumps.

H — The half carry flag is set to 1 if an add or subtract operation produces a carry into or borrow from bit 4 of the accumulator. Useful in BCD arithmetic. (Don't worry about this just now !!)

P/V — The parity and overflow flags share the same bit. In logical operations, even parity is shown by 1 and odd parity is shown by zero. In arithmetic operations, the flag is set to 1 if there was an overflow, or zero if not.

N The add/ subtract flag is set to zero if the last operation was an addition, or 1 if the last operation was a subtract.

C The carry flag is set to 1 if the last add instruction generated a carry, or the last subtract instruction generated a borrow.

The state of the flags after an operation is a useful guide to error conditions, and can often be used in conditional jump instructions. Many instructions can use flags to set conditions, for exapmle, CALL Z SUBROUTINE1 will call SUBROUTINE1 only if the zero flag is set. JP P, LABEL1 will transfer control to the address pointed to by LABEL1 only if the sign flag is set (ie there is a negative value in A). It is dangerous to draw too many parallels between these instructions and the conditional jumps performed in high level languages, as they are all unique, and there is no one to one correspondence.

16-bit values such as numbers larger than 127 (1111111 binary) and addresses of locations in memory (such as 3f80H from above) are stored in combinations of registers when necessary, thus you could store 3F80H in BC (registers B and C combined). The value would be stored as 80 3F (The high order byte *after* the low order byte). This is done to suit the Zilog electronics, and does not matter too much unless you subsequently refer to the wrong half of the register by mistake !

Thus the instruction LD BC, 3F80H would load the sixteen bit value into this register pair. You can use BC, DE, HL, IX and IY in this way. Having looked at the microprocessor, lets look at Z80 assembly language in more detail.

## Z80 Assembly language

Like most assembly languages, Z80 assembly language has the following features :

### a. **Mnemonic codes for operations**

These codes correspond on a one to one basis with the codes for operations already present in the processor. For every machine language instruction, there is a corresponding mnemonic. For example :

| **Mnemonic** (assembly language) | **Hex equivalent** (machine language) | **Action** |
|---|---|---|
| LD (HL),C | 71 | Load into the memory location (whose address is stored in the HL register pair) the contents of the C register |
| CONTENTS OF HL : (Remains unchanged) | 32A4 | (A hex address) |
| CONTENTS OF C : (Remains unchanged) | 4D | (A hex value) |
| CONTENTS OF 32A4 : | (initially) 00<br>(finally ) 4D | |

b. **Symbolic addresses**

The address of an instruction or data value in machine language is expressed by a binary, octal or hexadecimal code number. This can cause a great deal of inconvenience when attempting to locate a program in a new area of memory, or perform an operation on an indexed address. It is possible to use labels in a program to point to the start of a subroutine, so that you don't have to know the exact address of the byte in question. For example :

```
START:
```

is a label which marks the first byte of a routine called START. The exact address will be determined by the assembler during its operation. In BASIC programs, the line numbers are similar, in that they provide you with a labelled point in the program to jump to (eg GOSUB 1000)

c. **Automatic data conversion**

If the value of an item of data is to be used in a program, it must be expressed as a data type consistent with its fellow operands. (ie binary, octal, Hexadecimal) Assembly languages allow this, usually permitting the entry of denary numbers and converting them where necessary. It is also possible to use strings of characters instead of their hex codes.
For example:
The string :

```
Press<ESC>tocontinue..
5072657373203C4553433E20746F20636F6E74696E7565202E2E
```

Is far more comprehensible than the hex values of the ASCII codes beneath it ! Try to work out the denary equivalents of the hex codes !

d. **Directives**

In addition to instructions which correspond to their machine language equivalents, assembly languages usually have higher level instructions which have no equivalent in machine language.

For example :

```
ORG 100H
```

Is a directive stating that the program will start execution at the hex address 0100 in memory. (This is the standard place for CP/M programs to start by the way.)

Other directives possible in ZASM include the following :

| | |
|---|---|
| *INCLUDE | include another file of text in the program at this point. (Useful for including standard subroutines) |
| *HEADING | print a heading on each page of the listing file. |
| *PRINT message | prints whatever 'message' is during assembly. |

e. **Macros**

A macro or macro-instruction is a single instruction which represents a group of instructions. During assembly, whenever the macro instruction is encountered, the assembler puts all the instructions in the group into the program at that point. In this way, you don't have to type a commonly used sequence of instructions over and over if it is used many times. The assembler slots the group into the right place when needed.

The job of the assembler is to :

a. translate mnemonics into machine code
b. translate symbolic addresses into machine addresses
c. include the necessary linkages for closed subroutines
d. insert appropriate machine code for macros
e. allocate storage
f. detect invalid source language instructions
g. produce the object program on the requested medium
h. produce a printed listing of the source and object programs with comments

**An example program**

Having filled in some necessary background, the next step is to get to work on an actual program. Let's suppose we wish to write a very simple program to achieve nothing more than printing a message on the screen, having perhaps cleared the screen first.
In BASIC this program might look a little like this :

```
10 REM Program DEMO.BAS
20 PUT 31              :REM Clear the screen
30 PRINT "This is my first program in BASIC "
40 END
```

In assembly language we can create a program which will do exactly the same things, but will not need the BASIC interpreter to be running. First, we must design our program, then type it into a text-file. The text-file will be called DEMO.ZSM This is the default file-type required by ZASM the Z80 Assembler program. Well, what commands can we use to achieve the desired ends ? The answer to that, like everything in assembly language, is initially very confusing ! There are three ways to achieve the aims of our program.

1. We could design routines from scratch which will print a message stored at a given string of locations in memory. This would be very tiresome indeed, and is not a practical approach, unless you are a masochist.

2. We could use the routines kindly built into the Firmware of the computer by RML, called EMT calls (stands for Emulated Monitor Trap calls for some obscure reason !). This would be a good and efficient approach, *but* our program would then only work on RML machines, and we couldn't transport it to other CP/M machines (not that anyone will be interested in a program which just prints a message, but its the principle of the thing !)

3. We could use the routines which can be called from CP/M itself, in which case, our program would run on any CP/M machine.

I will show you both of the easy programs, and if you are very keen you can work out how to do it the other way on your own !

### Method One : EMT calls

Essentially, we have to output to the screen two things: first the control character 31 which will clear the screen. leaving the cursor at the top, and then a string of characters which makes up our message. This string must be put into memory and we must know the address where it starts. Luckily for us, ZASM will sort this out, so we can just refer to it as 'MESSAGE:' (very much like using MESSAGE$ in BASIC as a string variable). So far so good. Now there are two routines which RML supply built into the computer, (EMT calls) which we will use here. They are the routines OUTC (which puts the character currently in register A onto the screen) which we can use to do the equivalent of PUT 31 in BASIC ; and MSG which prints a message on the screen, if the address of the first byte of the message string is put into register pair HL. (The message string must also end with a special character – 0FFHex). The routines both have code numbers, and it is these that we use to call them. Having put the right values into the appropriate registers, we use the instruction :

```
        EMT 1       (for OUTC)
or      EMT 23      (for MSG)
```

That queasy feeling you will now be getting comes to us all, so sit tight and don't panic ! The EMT routines are fully documented in the Firmware Reference Manual mentioned earlier, and that is where I got the codes from. [OK – let's get to work. Type the following into a text file called DEMO.ZSM using TXED or WORDSTAR. (Everything from ORG to END START) :

```
        ORG  100H          ;The address where the
                           ;program will start
;Main program starts here

START:  LD   A,31          ;Put the code 31 into
                           ;reg. A ready for clearing
                           ;the screen
        EMT  1             ;Output 31 to the screen

        LD   HL,MESSAGE    ;Put the address of the first
                           ;byte of the message into HL
        EMT  23            ;Print the message on the
                           ;screen

        JP   0             ;Return to CP/M

;Define the message string

MESSAGE: DEFM      'This is my first assembly language program'
         DEFB      13,10,0FFH

;ending with a carriage return (13) and line feed (10)
;and finally the terminator 0FF Hex

         END       START
```

Notice the heavy use of comments (preceded by semi-colons) – ZASM ignores everything on a line after a semi-colon, (just like a REM in BASIC). There are no line numbers !!! The program is laid out so that it is easy to read and follow, with white space used freely, and all instructions kept in the same column, separated from their operands (the things they operate on) which are in another column. Labels such as START: and MESSAGE: at the leftmost side. Notice also that END is the very last statement, and does *not* denote the end of the program, which is actually the return to CP/M.

Assuming that went OK, and you now have a file called DEMO.ZSM containing the above program, the next step is to ASSEMBLE this source code to produce an intermediate file which will be called DEMO.HEX. Assuming you have the assembler program ZASM.COM on the same drive as the source code file, just type :

```
ZASM  DEMO    <RETURN>
```

*Don't* type the .ZSM as Zasm uses the filetype in a special way, and you will get a drive select error if you do ! (Look at the Zasm manual, page 1.2 onwards for the details). If there are any errors, just use TXED or WORDSTAR to edit the DEMO.ZSM file and correct them, then assemble the file again.

You should by now have a file called DEMO.HEX which looks like this if you TYPE it from CP/M :

**DEMO.HEX**

```
:100100003E1FF701210C01F717C300005468697303
:1001100020697320D6D792066697273742061737 2E
:10012000656D626C79206C616E6775616765 2070C2
:09013000726F6772616D0D0AFF28
:00010000FF
```

We now have to convert this to an executable binary file, called DEMO.COM, and to do this we need the CP/M utility program LOAD.COM, which you will find on your CP/M system disc. If it is on the same drive as DEMO.HEX, just type :

```
LOAD DEMO    <RETURN>
```

and you will see some mystical figures on the screen. You should now find that you have the files :

DEMO.ZSM (The original text file – the SOURCE CODE)
DEMO.HEX (The intermediate file)
DEMO.COM (The executable program – OBJECT CODE)

You execute the program like any other .COM program by typing its name (DEMO) from CP/M. Hopefully it will perform the same as it does on my system, but if you have an old 380Z, (COS 3.0) there may be differences.

## Method Two : CP/M function calls

Here is the same program in its "transportable" guise. I have altered it purely cosmetically and made it more readable by using lots of lower case, and (gulp!) the dreaded equates ! There's a list of these at the beginning of the program. (For example : CR equ 13 ) Many people find these disturbing at first, because they appear to have no obvious purpose. All they do is to inform the assembler that instead of using, say 13 throughout the program, you have used the initials CR, and it must *replace textually* all instances of CR with the value 13 before doing anything else. You can see from the listing that a carriage return (CR) is used, and you don't have to remember that 13 is the ASCII code for this character.

Equates can also be used in long programs to declare the values of constants only once (eg MAX equ 100), thus if the value of MAX has to be changed, you don't have to change it physically on the twenty lines in which you have used it, only once on the equate line. The assembler will replace all occurrences of MAX with whatever value you say it must have.

Now CP/M has a list of function codes (all fully documented in the CP/M and CP/NET Programmers Guide), and we will use two of them : Function 2 (Console output – called here Conout) ; and Function 9 (Print string – called here Print_string). Conout requires the ASCII character which is to be output to be put into register E , and the function number (2) into register C. When we have done this, we call the Basic Disc Operating System (BDOS) at location 0005 to let it know we want a CP/M function call executed. Print_string requires the address of the first byte of the string to be put into register pair DE, and the function number (9) into register C. Type in everything from ;Method Two to END START

### DEMO2.ZSM

```
;Method Two - using CP/M function calls :

CR              equ  13   ;ASCII code for carriage return
LF              equ  10   ;ASCII code for line feed
CLS             equ  31   ;ASCII code for clear screen character

BDOS            equ  0005 ;Address for BDOS calls
Warm_boot       equ  0    ;Address for warm boot operation

Conout          equ  2    ;CP/M Console output function code
Print_string    equ  9    ;CP/M Print string function code

                ORG  100H ;Address for start of program

;Main program starts here
Start:          Call Clear_screen   ;Subroutine defined below

                ld   de,Message     ;Put the address of the
                                    ;message string into DE
                ld   c,Print_string ;Put the CP/M function code
                                    ;for printing a string into
                                    ;register C
                Call BDOS           ;Tell bdos we want a function
                                    ;call
                jp   warm_boot      ;Return to CP/M with a warm
                                    ;boot operation
```

```
Clear_screen:  ld   e,CLS      ;Put 31 into register E
               ld   c,Conout   ;Put CP/M Console output code into
                               ;register C
               Call  BDOS      ;Tell bdos we want a function call

               Ret             ;Return to the calling part of
                               ;the program

Message:       defm       'My second assembly '
               defm       'language program !'
               defb       CR,LF,'$'

               END        Start
```

Notice that I also sneaked in a subroutine (Clear_screen:) which is not really necessary, since it is only called once! In a big program , of course, such a routine might be used many times, so it would be justified.

Type the file in again, and assemble and LOAD it as before. You should find that the program runs exactly like the first one, but don't forget – this one is transportable to other CP/M machines, and is strictly speaking, the correct way to write programs.

# Using Front Panel to Change Automatic Filetyping

*by John Morris (Suffolk College)*

A chance remark by a colleague led to this train of thought:- "Wouldn't it be nice if you could tell which version of BASIC a program needs just by looking at its name. Perhaps BASIC6 programs could end in BA6, and SBAS programs could end in BSS."

And then, later, another thought (it was a vintage month):- "Wouldn't it be nice if not *all* word processors used BAK to identify their backup files."

So can we do anything about it? Well yes – and it isn't too difficult to do. Much more difficult to explain to a unknown audience – but here goes. Here, in outline, is my top level design for the process:-

1. Determine size of program and load it into memory.
2. Use front panel to change the automatic filename extension.
3. Save the new version.
4. Check the new version.

Before I go off into the detail, a few explanations to make sure you know what I mean by certain bits of jargon. Using CP/M most filenames have two parts separated by a full stop. The second part, up to three characters long, is called the filename extension or the filename type. Thus all CP/M system programs have the extension COM, all BASIC programs have the filename type BAS (until now), etc etc. Many pieces of software add these filename extensions to the filename automatically (e.g. BASIC or will only work with files named with the correct type (e.g. ZASM or TCL PASCAL etc).

To be more explicit:-

BASIC will automatically save programs with type BAS and will only look for files (Load or Dir) with type BAS. (Both can be manually overridden.)

WORDSTAR will automatically produce backup files with type BAK.

LOGO procedures will be saved and loaded (by LOGO) from a file with type LGP.

WRITE (about which more later) automatically produces files with type TXT and backup files with type BAK.

To produce and use these filetypes automatically, a piece of software must have the characters embedded within itself somewhere. So within BASIC there must be the characters BAS, within WORDSTAR there must be the characters BAK, within LOGO must be the characters LGP. If we can find and change these triplets of characters the software will use different filename types. The first piece of software I modified was WRITE. I will explain exactly how I did that, and in tandem explain how you can apply the same trick to other pieces of software.

WRITE is an excellent educational word processing package available at reasonable cost from Oxfordshire C.C. (This is a plug – completely unsolicited.) Besides being delightfully easy and flexible to use, WRITE allows you (nay, almost invites you) to modify it to suit local customs. This I did. But then I kept losing my WRITE files amongst all the other files on our (at that time) rather overfull and chaotic discs. Not all TXT and BAK files would work with WRITE.

The manual tells you that to modify WRITE, you must first run the program, open a document but without typing anything into it, use the commands to change the options (things like printer type, paper size, margins, tabs, justifying, screen display, page breaks, etc etc), check the size of the program (use the inbuilt values option, which also tells you which printer the program is configured for, and how many words in your document) and then exit the program (you *Quit* the document and then *Finish* the program). Now says the manual, type :-

```
SAVE nn NWRITE.COM<return>
```

where nn is the size of the program previously determined. This must mean that the WRITE program is actually sitting in memory with all the changes made incorporated within it. So I have achieved step 1 of my top level design.

Actually we need one more piece of information about the WRITE program, and, rather than make you jump through hoops, I will tell you what I think it is. My (and, I guess your) version of WRITE tells you that it is 73 pages long. Which means that it occupies memory from 0100 to 49FF – both numbers are in hex.

Other pieces of software might not be quite so accomodating. So I will tell you a method which will work with (almost) all other software. (I only put the almost in there to cover the case I haven't discovered – I don't mind you writing to tell me what I've done wrong, but I'll try and reduce the flow.) To be on the safe side prepare a disc with the software to be modified and DDT.COM on the same side. You will also need to use STAT. There should be enough space on the disc for a second copy of the software. (You will also need to use STAT.) Log onto this side of the disc and type:

```
DDT filename.COM<return>
```

You will see something like this

```
DDT VERS 2.2
NEXT  PC
%%%% 0100
-
```

where %%%% is a four digit hexadecimal number that tells you the size of the program *of which you must make a note* and where – is DDT's prompt. (For our BASICSG2 %%%% was 5380.) Type:

```
GO<return>
```

and you should get back to the CP/M prompt (A> or B> or. . .).

The software is now loaded into memory. It occupies the memory from location 0100 to one location before %%%% – both numbers are in hex. Later on we will need to know the size of the program in pages – and this number will need to be in decimal. There are two ways to calculate this. The first method is exact but a little daunting perhaps. The second method is a little cruder but easier on the arithmetic.

To use the first method we need the number %%%% mentioned above. Take the first digit of %%%%, multiply it by 16, add the second digit and, *only if* the last two digits are *both* 0, subtract 1. You will need to remember that some hex digits are numbers – A is worth 10, B is worth 11, C 12, D 13, E 14 and F is worth 15.

```
E.G.    %%%%
        1234 —> 1 * 16 +  2 =  18
        3B04 —> 3 * 16 +  4 =  59
        5380 —> 5 * 16 +  3 =  83
        6F00 —> 6 * 16 + 15 - 1 =  111
```

If you are daunted by all that we can use the second method. Before you get into the DDT bit above use STAT to find the size of your program. E.g. type

```
STAT BASICSG2.COM<return>
```

You will see

```
Recs  Bytes  Ext  Acc
 165    24k    1  R/O A:(BASICSG2.COM)
```

Take the Recs number and divide it by 2, rounding any halves up. I.e. from the above, the result will be 83, and in this case the two answers are the same.

If you've done all that you are now sitting at a machine with the software loaded into memory, and you know how large the software is (in pages, in decimal), and you know exactly where in memory the program is (in hex). So now to step 2.

In other words, Front Panel. Hold down the CTRL button and tap F. Impressive eh? Of all the numbers on the display, on this occasion we are only interested in the bottom eight lines which will look like something this:

```
00F4 6B    00FC 44    0104 11    010C C3
00F5 C8    00FD 00    0105 09    010D CF
00F6 00    00FE 3F    0106 C3    010E 15
00F7 B2    00FF C2    0107 20    010F C3
00F8 00   >0100 C3<   0108 09    0110 F7
00F9 BA    0101 12    0109 C3    0111 4D
00FA 4C    0102 51    010A B0    0112 C3
00FB C8    0103 C3    010B 10    0113 2B
```

What we are looking at is a display of the addresses and contents of thirty two locations of the machines memory. One location is highlighted with arrows – this is the location that we can directly change.

The four digit hex numbers are the addresses of the locations, and the two digit hex numbers are one form of representing the bit pattern stored in those locations.

Front panel has a wide range of commands that can be used to do many clever (and simple) things. I will not attempt to use them all or even to explain all the intricate details of using them. Interested readers can look them up in the relevant manual. For the purpose of this article just follow the steps enumerated below. In some cases there are minor differences if you are using a 380Z rather than a 480Z – I will enclose the 380Z version in stars (*)

1. Make sure that the arrows point to location 0 10 0 – the beginning of the program. If they don't, type

```
M Ø  1Ø  Ø  <return>
```

As soon as you type M a > will appear so the screen will actually look like this at the bottom :

```
!M>Ø  1Ø  Ø!
```

2. If you have a 480Z, type

```
T
```

This changes the display so that each different bit pattern in the locations are represented by different characters – you will recognize letters, digits, and graphics characters. If there is any text embedded in your program this will make it easier to recognize it.

*380Z users will have to find an ASCII-to-hex conversion table and have it close to hand. The last one I saw was in a Read Using, but which issue was it?*

3. To find the filename extension XYZ, type

```
G
"XYZ<return>
<return>
```

*To find the filename extension XYZ, first look up the characters X Y and Z in the conversion table – in this particular case the equivalents are 58 59 and 5A. Then type:

```
G
58<return>
59<return>
5A<return>
<return>        *
```

4. The screen will now have changed. The location indicated by the arrows will be different and the contents of it will be the first letter or its code of the filename extension we seek.

5. To insert the new filename extension PQR type

```
"PQR<return>
```

*To insert the new filename extension PQR, look up the letters P Q and R – the equivalents are 50 51 and 52 – and then type:

```
50  <return>
51<return>
52<return>      *
```

6. The original filename extension may appear more than once in the program. To check this type:

```
N
```

The display will change again. Look at the address of the highlighted location. If the address is still within the program (remember that we worked out the boundaries of the program earlier) repeat from step 5.

If there were two extensions that you wished to change in the one program go back to step 1.

And that's nearly it.

We now must save the modified program. At this stage *I strongly recommend* that you give your new, modified program a *different* name to the original. After all you might have made a mistake. Type

```
SAVE nn newname.COM<return>
```

where nn is the size of the original program in pages in decimal.

You must now test your modified program. When you are satisfied that everything works the way that you think it should, you can save your original program on a disc which you keep in a secure place. Rename the modified program to the original name, make it read only (and maybe SYS) and go and make yourself a peanut butter and jelly sandwich.

# Using RUN.COM from any User Area

*by Neil Howie*

The RML network documentation for version 2.2 on states that the parallel download program for non-silicon-disc systems, RUN.COM, restricts users to working on user 0.

Whilst it is true that it is not possible to issue a USER command in the command line which will change the user number of all stations once booted, it *is* possible to run a program which has the same effect. Such a program may also change the logged on drive, clear the screen of text and HRG and any other tidying-up we may require, for example the program given in the article "Clearing Screens" by *John Graham-Cumming* in RU 14. Once implemented, programs from user 0 with the SYS attribute set or programs from the new user number, may be loaded and run at will.

Another such program, which takes the required new drive and user number as parameters in the command line, is listed in source code form below. This may be assembled with ZASM and turned into a command level program using the LOAD utility in the usual way. A user who wishes to change to user 5, drive B would type:

```
A>LOGIN  5<return>
```

The space after LOGIN is obligatory, any thereafter are ignored. The parameter order must be first drive then user number.

To use this program with RUN.COM you could load and run, on stations 6 to 12 inclusive, a BASIC program called GRAPH.MAT saved on drive C, user 7 when BASICSG2.COM is on drive B, user 0 and RUN.COM and LOGIN.COM are on drive A, user 0 with:

```
A>RUN [06-12] LOGIN C7!B:BASICSG2 GRAPH.MAT
```

It can be used in a similar manner with the silicon disc program BOOT.COM and it may be possible to make it an initial option in a menu driven 'front end' network management system such as ACCESS, though this has not yet been tried.

```
;LOGIN sets or requests drive and user number
;Version 1.1          N. Howie, January 1986

        bdos    equ     5
        org     100h
        ld      d,0ffh      ;set flag for 'no user number'
        ld      c,0         ;zero flag for 'no first digit'
        ld      hl,80h      ;command line buffer address
        ld      b,(hl)      ;first char is number of bytes
```

```
nexta:  inc     hl
        ld      a,(hl)      ;scan command line
        cp      20h         ;ignoring spaces.
        jr      nz,drfnd    ;first character in register a
        djnz    nexta
        jr      error
;N.B. CP/M converts lower case to upper case
drfnd:  cp      '?'         ;query or change?
        jr      z,info

        cp      81          ;change so check 'A' to 'P'
        jr      nc,error
        ld      (drive),a   ;copy it to message

        sub     65          ;and convert to a=0, b=1 etc
        jr      c,error     ;if legal
        ld      e,a
        dec     b
nextb:  inc     hl
        ld      a,(hl)      ;now look for user number
        cp      20h         ;ignoring spaces.
        jr      z,nousn

unfnd:  cp      3ah         ;number in reg a-check '0' to '9'
        jr      nc,error
        sub     30h         ;converting to 0 to 9 if legal
        jr      c,error
        ld      d,a         ;save digit (clearing 0ffh flag)
        ld      a,c         ;check for digit already found
        or      a
        jr      nz,tens     ;if so check for ten
        ld      c,d         ;else save this digit
nousn:  djnz    nextb       ;and look for second

        inc     b
        ld      a,d         ;no more characters so check
        cp      0ffh        ;to see if user number found
        jr      z,error
        ld      a,c
        or      a           ;clear flags
        jr      login

tens:   cp      1           ;check first digit
        jr      nz,error
        ld      a,d         ;if OK the new one should be
        cp      6           ;less than 6
        jr      nc,error
        add     a,10        ;in which case accept and add ten.
login:  ld      c,a         ;save user number for later
        rla                 ;shift into first nibble
        rla
        rla
        rla
        add     a,e         ;put drive in last nibble
        ld      (4),a       ;and save the byte.
        jr      testb       ;any more characters in command
nextc:  inc     hl          ;line buffer?
        ld      a,(hl)      ;if so they should be
        cp      20h         ;spaces!
        jr      nz,error
testb:  djnz    nextc
        ld      a,c         ;recall user number
        ld      de,2020h    ;move spaces
        ld      (user),de   ;to message
        ld      hl,user
        cp      10          ;check for user >= 10
        jr      c,noten
        sub     10          ;get units
        ld      (hl),'1'    ;and set up tens if it is
        inc     hl
```

```
noten:  add     48
        ld      (hl),a      ;move units to message
        ld      de,ack
        jr      msgout

error:  ld      de,errmsg   ;or pick up error message
msgout: ld      c,9
        call    5           ;and output it
        rst     0           ;then perfor warm start

;option to display current drive and user number

info:   ld      a,(4)       ;fetch drive/user number
        and     0fh
        add     65
        ld      (drive),a   ;move drive to message
        ld      a,(4)
        and     0f0h
        rra
        rra
        rra
        rra
        ld      c,a         ;and save user number
        jr      testb

errmsg: 'Login Version 1.1',13
        13,'Usage:  LOGIN B 12',13
        '   or   LOGIN ?',13,'$'

ack:    'Logged on to drive '
drive:  defs    1
        ', user '
user:   defs    2
        13,'$'
        end
```

Footnote: I wanted to include a check that the drive being accessed was available, to give an intelligent message instead of the dreaded NDOS error 05. After refering to RML's *CP/M and CP/NET Programmer's Guide* PN12048 I have tried looking at the server configuation table with function 71 which is supposed to tell me the available drives, and poked around the BDOS entry point subroutine SELDSK which is supposed to return HL=0 if there is an error in trying to return a non-existent drive, but all to no avail. Can anyone help?

Oakwood Park Grammar School, Maidstone — January, 1986

# Patching TXEDE V 4.2 I for the Walters WM2000 printer

*by Ray Lovell*

I refer to a previous article on this subject by Haydon Charles (see *Read Using 16*). This article mentioned it was only possible to emphasize whole lines on the WM2000 printer from TXED and not individual words on a line – this is not the case.

To make the reprint option work correctly a 'force-out' (ESC CR) is required before the reprint code (ESC 'R') is sent. Unfortunately ESC CR cannot be used because TXED interprets a carriage return as a carriage return followed by a line feed, so CR+128 (8DH) is used instead to overcome this problem. The ".BC" command can then be used to perform the reprint function providing the Bold Face Sequence ("BFS") is patched with the following code: 1B, 8D, 1B, 52, FE, 1B, 8D, 1B, 52, FF. The "BFS" can be patched via the Front Panel or by using ZASM and DDT. The following assembler file may be of use if you wish to change other values in addition to "BFS" etc. or even to add a new immediate mode command (CTRL L) which moves the cursor to the end of a line.

```
*H      WALTERS.ZSM

; TXED V 4.2 I with Walters codes
; Last modified on 03/06/86
        BS      EQU     8
        FF      EQU     12
        CR      EQU     13
        ESC     EQU     27
        ORG     01B5H
CTRL_L: DEFB    FF
        DEFW    0207H
        ORG     0207H
EOL:    DEFB    'C','L','-','C'
        DEFB    -1
        ORG     1E36H
PCS:    DEFM    'Walters codes'
        ORG     2175H
.LL:    DEFB    72
.IN:    DEFB    0
        DEFB    0
:       DEFB    3
.LS:    DEFB    1
        DEFB    1
.PL:    DEFB    72
.TS:    DEFB    3
.BS:    DEFB    3
.HS:    DEFB    0
.LM:    DEFB    6
        DEFW    1
        DEFB    0
```

```
            DEFB    0
.UC:        DEFB    0
            DEFB    0
            DEFB    0
.FS:        DEFB    0
.NC:        DEFB    '£'
.BC:        DEFB    0
            DEFB    0
            DEFB    0
.E1:        DEFB    0
.E2:        DEFB    0
.E3:        DEFB    0
.E4:        DEFB    0
.E5:        DEFB    0
PULS:       DEFW    21A2H
PBFS:       DEFW    21AAH
PPFS:       DEFW    21BAH
PESC1:      DEFW    21C2H
PESC2:      DEFW    21C9H
PESC3:      DEFW    21D0H
PESC4:      DEFW    21D7H
PESC5:      DEFW    21DEH
FIX:        DEFB    ' '
ULS:        DEFB    ESC,'U',-2,ESC
            DEFB    'U',-1,0,0
BFS:        DEFB    ESC,CR'128,ESC,'R'
            DEFB    -2,ESC,CR'128,ESC
            DEFB    'R',-1,0,0
            DEFB    0,0,0,0
PFS:        DEFB    -1,0,0,0
            DEFB    0,0,0,0
ESC1:       DEFB    ESC,-1,0,0
            DEFB    0,0,0
ESC2:       DEFB    -1,0,0,0
            DEFB    0,0,0
ESC3:       DEFB    -1,0,0,0
            DEFB    0,0,0
ESC4:       DEFB    BS,-1,0,0
            DEFB    0,0,0
ESC5:       DEFB    -1,0,0,0
            DEFB    0,0,0
            END
```

All you have to do is to create an ASCII file of the above called WALTERS.ZSM from TXED or WORDSTAR and then use ZASM followed by DDT – details follow. (TXEDE, ZASM and DDT need to be on the same disk, drive A in this example.)

```
TXED WALTERS.ZSM  <RETURN>          type in assembler file and save it!

ZASM WALTERS.AAA  <RETURN>          wait!

DDT TXEDE.COM    <RETURN>
IWALTERS.HEX         <RETURN>
R                    <RETURN>
CTRL C
SAVE 44 TXEDW.COM <RETURN>
```

Royal Grammar School, High Wycombe

# Patching WSMSGS.OVR

*by Neil Howie*

Thank you, John Morris, for your notes on getting the best from WORDSTAR with my Epson FX80. I have gone on to patch the file WSMSGS.OVR so that the changes are now documented in the print menu. The fact that this file (or any other) is not a .COM program does not prevent it being loaded using DDT, played around with and then saved using the CP/M SAVE command. Therefore the method of incorporating the new headings, once one has poked around in memory to find out where they must go to replace the old ones, is just the same as described for patching WORDSTAR itself.

I prepared a file called MSGPATCH.ZSM as follows:

```
        org     1536h
        defm    'full underlining'

        org     158eh
        defm    'chars'

        org     15a1h
        defm    'chars'

        org     173bh
        defm    'chars'

        org     178ah
        defm    'chars'

        org     181eh
        defm    ' Y Full Underlining  '

        org     186eh
        defm    ' Q Condensed  W Wide '

        org     18beh
        defm    ' E Elite   R Restore '
```

The first three entries only alter the self-documentation that appears before the menu and are non-essential. The other five modify the displayed menu itself, and the last three must contain exactly twenty-one characters between the single quotes.

The next stage was to assemble this: (what one types is underlined)

```
ZASM MSGPATCH
```

hopefully getting:

```
ZASM  sign on and copyright message
    .... bytes free
    No errors detected
```

then to merge the two files

```
A>DDT WSMSGS.COM

    DDT VERS 2.2
    NEXT  PC
    6E00 0100
    -IMSGPATCH.HEX

    -R
    NEXT  PC
    6E00 1536
    -G0
```

and to replace the old with the new

```
A>REN WSMSGS.OLD=WSMSGS.OVR

A>SAVE 109 WSMSGS.OVR
```

To test out the new version, type

```
A>WS DUMMY
```

and when in the text entry mode with the help level set to 2 or 3, type control P and wait for the print menu. Note any faults in the display then exit with control KQ followed by X. If there are faults, check the contents of the .OLD version at the addresses given in the above code by reloading with DDT and after the '-' prompt typing D1530,18cf (toggling autopaging as necessary with control A).

O.P.G.S. January, 1985

External Winchesters for Nimbus July 1986

# Whittle Systems

## 20 & 40 Megabyte external Winchesters

**SAVE up to £1501** On an external Winchester (WS40 XW cf XW40 @ educational prices)

Is your Network running out of disk space?
Then why not consider adding one of our high performance external 20 or 40 magabyte Nimbus Winchesters as an additional Winchester, giving you that vital extra capacity, higher performance, and saving you money?

### WS20XW

- Full 20 Megabytes capacity
- Compatible with MSDOS, MSNET,CHAIN64
- Twice as fast as other 16Mb versions
- Over 160k bytes/sec
- Simple 1 – 2 – 3 installation
- 12 Months warranty

**Down by £70**

**£925 education**
**£1200 commercial**

### WS20FW

A WS20XW with a 5.25 inch floppy disk drive allows you to Read/Write

- IBM PC MSDOS disks
- 480Z CP/M disks

**New**

**£1115 education**
**£1395 commercial**

### WS40XW

- Full 40 Megabytes capacity
- Partitioned as two drives
- Compatible with MSDOS, MSNET, CHAIN64
- Over 160k bytes/sec
- Simple 1 – 2 – 3 installation
- 12 Months warranty

**Down £200**

**£1695 education**
**£2225 commercial**

*All prices exclude P & P and Vat.*

**9, Herschel Crescent, Littlemore, Oxford OX4 3TS** Tel 0865 711956

# Redefining the 380Z character set

*by N.P. Schiettecatte.*

Many people have difficulties in making their graphical programs (eg Basic, Fortran, Assembly language) fast due to the fact that lines, blocks,'creatures from the depth' and other graphical objects in HRG take a relatively long time to draw. This is due to the fact that on the RML 380Z and 480Z, the HRG screen is difficult to access and modify. An alternative is to use the character set for graphic (ie CHR$(number) ). This does not produce a very good resolution and is extremly chunky; the user is thus restricted in resolution and a choice of usable characters.

The way to make graphics fast is to redefine the character set, which can be done to characters 128 to 255. (*Only on the 380Z*!!) Once redefined, the characters are of an exeptional resolution, ie the pixels used to create the character are smaller than the high resolution pixels. Unfortunately, if one defines a man, he can only be made to move in blocks and he cannot be drawn in colour. The advantage is that the man will move a lot faster that if he was plotted and unplotted in HRG. This is because the computer does not have to access and change the HRG screen, pokes the number of the character into the right place of the memory where the data for the text screen is kept, thus the character appears on the screen.

The pixels of the characters are set in an eight (across) by ten (down) matrix. To draw a man, the blocks of the grid are filled to form the shape of a man. To calculate the data for the character, you draw numbers on top of the grid, starting form the top-right with the number one; for every block to the left, the number is multiplied by two. For each row, you add the numbers of all the blocks that are filled, so that all the rows end up with a number. An example is shown .

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | *** | *** | | | | 16 + 8 | = 24 |
| | | | *** | *** | | | | 16 + 8 | = 24 |
| | | *** | *** | *** | *** | | | 32 + 16 + 8 + 4 | = 60 |
| | *** | | *** | *** | | *** | | 64 + 16 + 8 + 2 | = 90 |
| | *** | | *** | *** | | *** | | 64 + 16 + 8 + 2 | = 90 |
| | | | *** | *** | | | | 16 + 8 | = 24 |
| | | *** | | | *** | | | 32 + 4 | = 36 |
| | *** | | | | *** | | | 64 + 2 | = 66 |
| | *** | | | | *** | | | 64 + 2 | = 66 |
| *** | | | | | | *** | | 128 + 1 | = 129 |

## Redefining a character.

```
10 PUT 27,60,128          : This sets up character 128 to be redefined.
20 FOR J=1 TO 10          : Loop for ten lines of pixels.
30 READ A                 : Value of 'A' defines the line of 8 pixels.
40 PUT A
50 NEXT J
60 PUT 0,0                : Two zeros follow the last item of data.
                          : These zeros terminate the procedure.

70 DATA 24,24,60,90,90,24,36,66,66,129
```

## Redefining a character in assembly language.

```
        ORG     256         ;Sets the origin of the program at 256.
                            ;(N.B. this is the same as 100H)
        LD      HL,TABLE    ;Loads HL with pointer to the data.
        LD      A,27        ;Loads A with the number 27 (Escape).
        EMT     OUTC        ;'PUT' A.
        LD      A,60        ;loads A with the number 60 (Sequence
                            ;number for redefining a number).
        EMT     OUTC        ;'PUT' A.
        LD      A,128       ;loads A with the number 128.
        EMT     OUTC        ;'PUT' A
                            ;This defines character 128 to be redefined.
        LD      B,10        ;Get B ready for a 'for-next' loop of ten.
LOOP:   LD      A,(HL)      ;Loads A with data pointed to by (HL).
        INC     HL          ;Brings up the data pointer to the next bit of data.
        EMT     OUTC        ;'PUT' A ('A' is the next data value).
        DJNZ    LOOP        ;Next loop.
        LD      A,255       ;Loads A with number 255.
        EMT     OUTC        ;'PUT' A.
        LD      A,0         ;Loads A with number 0.
        EMT     OUTC        ;'PUT' A.
                            ;These last four lines are the equivalent of PUT 255,0.
                            ;This terminates the procedure.

TABLE:  DEFM    24,24,60,90,90,24,36,66,66,129
```

You will probally notice that unlike the BASIC version, this does not end with PUT 0,0 but with PUT 255,0; in fact BASIC is the only language for the RML 380Z that ends the redefining sequence with PUT 0,0 . All the other languages end it with PUT 255,0.

**The Program**

There is a game provided in the following listing to demonstrate this routine. To save typing it out, get a copy from the *User Group Library*.

(*Only if Nicholas has sent it in and claimed his free library volume!* – Editor)

Here are the instructions on how to play the game:-

The player controls a see-saw with two men on it and can move it left with the key '<' and right with the key '>' (Shift key not regarded).

The object of the game is to bounce the men ( One at the time ) and catch birds that fly accross the screen; the men can bounce on the girders that are situated about the screen and can bounce off the 'force fields' at the sides of the screen.

The *Jumping Ability* is set by pressing keys 1 to 9 at the top of the keyboard. The Jumping Ability controls the the height and hence how far the jump goes horizontally and vertically. With a jumping ability of 9 the man will bounce to the top of the screen and also go a long way while a jumping ability of 1 he will hardly bounce or move any where at all.

So it is possible to control the direction and speed of the man by changing the jumping ability.

The seesaw must be positioned so that the man in flight lands on the free end of seesaw. If the seesaw is in the wrong position, the man will fly off the bottom of the screen and the player will lose one of his three lifes. Once a life is lost, the screen clears itself and a new screen is plotted and the player starts again but with one life less.

When the player has lost all his lives, the game ends. If he has done well enough in the game, he will be able to enter his name into the high-score board. I hope that this routine to redefine characters will come in useful to all *Read Using* readers who have 380Z computers.

Variables used in the program.

| | |
|---|---|
| HS(1-12) | :Varaible of high scores from 1 to 12. |
| HS$(1-12) | :This 'string' variable holds all the names of<br>:the people with high scores from 1 to 12. |
| BX(1-9) | :X Co-ordinates of the 9 birds. |
| BY(1-9) | :Y Co-ordinates of the 9 birds. |
| BI(1-9) | :Direction in which birds 1 to 9 are travelling. |
| BQ(1-9) | :Which side of the screen a bird (from 1 to 9) start off if<br>:it flew off the side of the screen or if it was caught. |
| WW | :Which bird is being moved this go. It can be 1 to 9. |

| | |
|---|---|
| WT | :This changes flight pattern of the birds. If WT=1 then bird<br>:has it's wings up. If WT=2 then bird has it's wings down. |
| X | :The X co-ordinate of the seesaw. |
| OX | :The old X co-ordinate of the seesaw. |
| JX | :The X co-ordinate of the man in flight. |
| JY | :The Y co-ordinate of the man in flight. |
| XX | :The direction in which the man is flying on the horizontal plane. |
| YY | :The direction in which the man is flying<br>:on the vertincal plane. |
| IQ | :Which man is on the seesaw.<br>:If IQ=1 then man is on the left hand side of the seesaw.<br>:If IQ=2 then man is on the right hand side of the seesaw. |
| QI | :Which man is jumping.<br>:If QI=1 then the man is facing to the left.<br>:If QI=2 then the man is facing to the right. |
| TW | :Jumping ability. Controls speed of jumping character. |
| N(1-6) | :Each one (From 1 to 16) holds a value of 0 to 9.<br>:These construct the score in redefined characters. |
| PO: | :The number of lives the character has left. |
| SC | :The player's score. |
| A$(1-2) | :These strings are a group of redefined characters<br>:to look like a seesaw.<br>:A$(1)= A seesaw with a man on the left.<br>:A$(2)= A seesaw with a man on the right. |
| B$(1-2) | :These strings each hold a redefined picture of<br>:the man that is jumping.<br>:B$(1)= A man facing to the left.<br>:B$(2)= A man facing to the right. |
| C$(2,2) | :These contain the bird characters. C$(x,y)<br>:x defines the direction in which the bird is flying.<br>:y defines if the wings are up or down. |
| W$ | :This is the word "SCORE>=" in redefined characters. |
| Q$ | :This is word "JUMPING ABILITY>=" in redefined characters. |
| LV$ | :This is the word "LIVES>=" in redefined characters. |
| N$(0-9) | :These are characters redefined as numbers 0 to 9. |
| J | ) These are used in 'for-next' loops. |
| TU | ) |
| I | ) These three variables are use in finding random |
| O | ) numbers. |
| U | ) |

```
10 REM ********************************
20 REM ******** Flip - Flop *********
30 REM ************ By ***************
40 REM ***** N.P.Schiettecatte ******
50 REM ******* Douai School *********
60 REM ********************************
70 REM
80 REM ========================================
90 REM **** Clears screen and Dimensions arrays.
100 REM ========================================
110 PUT 12,27,"=0J"
120 CLEAR 2000
130 RANDOMIZE
140 DIM HS(12),HS$(12)
150 FOR J=1 TO 10: HS$(J)="NICKSOFT": HS(J)=550-(J*50): NEXT J: HS(0)=100000
160 GOTO 900
```

```
170 REM ======================================
180 REM ****        Start of game.        ****
190 REM ======================================
200 REM
210 REM **** MAIN LOOP
220 REM
230 REM ======================
240 REM **** Moving the birds.
250 REM ======================
260 WW=WW+1: IF WW>9 THEN WW=1  :REM ** W = The number of the bird to be moved.
270 PLOT BX(WW),BY(WW)," "      :REM ** Unplotting the bird.
280 IF WT=1 THEN WT=2 ELSE WT=1 :REM ** Changes the bird's wings from up
290                             :REM ** to down and vice versa.
300 BX(WW)=BX(WW)+BI(WW)        :REM ** Increases the bird's X co-ordinate
310                             :REM ** by adding it to it's variable containing
320                             :REM ** it's direction.
330 IF POINTS(BX(WW),BY(WW))=178 THEN I=INT(RND(1)*2)+1: IF I=1 THEN BY(WW)=BY(WW)+3 ELSE
BY(WW)=BY(WW)-3
340                             :REM ** Checks wether the bird has hit a
350                             :REM ** girder, if so then it moves either
360                             :REM ** one place up or one place down.
370 IF BX(WW)<6 THEN BX(WW)=71: BY(WW)=(INT(RND(1)*15)*3)+6
380 IF BX(WW)>71 THEN BX(WW)=10: BY(WW)=(INT(RND(1)*15)*3)+6
390                             :REM ** The last two lines checked wether the
400                             :REM ** bird has gone off the side, if so then
410                             :REM ** it reappears on the other side, but
420                             :REM ** different Y co-ordinate.
430 PLOT BX(WW),BY(WW),C$(BQ(WW),WT) :REM ** Plots bird.
440 REM ======================================
450 REM **** Getting' a key from the keyboard.
460 REM ======================================
470 G$=GET$(0)
480 OX=X                :REM ** Putting the original value of X
490                     :REM ** into variable OX.
500 REM ================================================
510 REM **** Deciding what to do when a key is pressed.
520 REM ================================================
530 IF G$="," THEN X=X-2: IF X<-8 THEN X=80
540 IF G$="." THEN X=X+2:IF X>82 THEN X=-6
550 IF G$>"0" AND G$=<"9" THEN TW=INT((ASC(G$)-44)/2.3)-0.3: PLOT 0,57,Q$+N$(ASC(G$)-48)
560 REM ================================================
570 REM **** Unplotting the characters involved,working out
580 REM **** where they move and replotting them.
590 REM ================================================
600 PLOT OX,0,"    "    :REM ** Unplotting the seesaw with variable OX.
610 PLOT X,0,A$(IQ)     :REM ** Plotting the seesaw with X.
620                     :REM ** OX is the old value of X and X is the
630                     :REM ** new value. By doing this, the  seesaw
640                     :REM ** is plotted straight after it is unplotted.
650                     :REM ** Hence there is a minimal  time  between
660                     :REM ** unplotting and plotting, and so the amount
670                     :REM ** of flickering is greatly reduced.
680 PLOT JX,JY," "
690 JX=JX+XX: YY=YY-0.2: JY=JY+YY: IF YY<-2.8 THEN YY=-2.8
700 IF JX<2 OR JX>76 THEN GOSUB 3070 :REM ** Checks whether the character
710                                   :REM ** has gone off the side of the
720                                   :REM ** screen.
730 IF JY<-4 THEN GOTO 2150 :REM ** Checks is the character has gone off
740                         :REM ** the bottom of the screen.
750 A=POINTS(JX,JY): IF A=178 THEN JY=JY-YY: YY=-YY: A=32
760                         :REM ** Checks if the character has hit a girder.
770 IF A>158 THEN GOTO 1920 :REM ** Checks if the character has caught a
780                         :REM ** bird.
790 IF A=32 OR A=128 THEN GOTO 840
800 IF JX<XI+X-2 OR JX>IX+X+2 THEN GOTO 840
810 IF IQ=1 THEN IQ=2: QI=1: XX=1: JX=X: YY=TW: IX=0: GOTO 840
820 IF IQ=2 THEN IQ=1: QI=2: XX=-1: JX=X+6: YY=TW: IX=6
830                         :REM ** Checks if the man has landed on the seesaw.
840 PLOT JX,JY,B$(QI)       :REM ** Plots man.
850 GOTO 260                :REM ** Goes to the beginning of the main loop.
```

```
860 REM ========================================
870 REM **** Reading data, and using it to define
880 REM **** low Resolution Characters.
890 REM ========================================
900 FOR J=129 TO 179     :REM **** Ready to redefine characters 129 to 179.
910 PUT 27,60,J          :REM **** This tells the 380Z to get ready to redefine
920 :REM **** character J'.
930 FOR U=1 TO 10        :REM **** The computer reads and redefines the ten
940 READ A               :REM **** vertical lines of the characters.
950 PUT A
960 NEXT U
970 PUT 0,0              :REM **** This terminates the redefining of the character.
980 NEXT J
990 REM ========================================
1000 REM **** Putting all the defined Characters
1010 REM **** into variables.
1020 REM ========================================
1030 A$(1)=CHR$(129)+CHR$(130)+CHR$(131)+CHR$(132)
1040 A$(2)=CHR$(133)+CHR$(134)+CHR$(135)+CHR$(136)
1050 B$(1)=CHR$(137): B$(2)=CHR$(138)
1060 FOR J=139 TO 153: Q$=Q$+CHR$(J): NEXT J: Q$=Q$+">="
1070 FOR J=154 TO 158: W$=W$+CHR$(J): NEXT J: W$=W$+">="
1080 C$(1,1)=CHR$(159): C$(1,2)=CHR$(160)
1090 C$(2,1)=CHR$(161): C$(2,2)=CHR$(162)
1100 FOR J=163 TO 172: N$(J-163)=CHR$(J): NEXT J
1110 FOR J=173 TO 177: LV$=LV$+CHR$(J): NEXT J
1120 REM ========================================
1130 REM **** defining the co-ordinates of the Birds
1140 REM **** and the  direction in  which they  are
1150 REM **** going.
1160 REM ========================================
1170 FOR J=1 TO 9
1180 I=INT(RND(1)*2)+1
1190 IF I=1 THEN BQ(J)=1:BI(J)=2
1200 IF I=2 THEN BQ(J)=2:BI(J)=-2
1210 BX(J)=(INT(RND(1)*30)*2)+10: BY(J)=(INT(RND(1)*16)*3)+3
1220 NEXT J
1230 REM ================================
1240 REM **** Giving values to variables.
1250 REM ================================
1260 IQ=1: XX=0.5: YY=3: X=8: P0=3
1270 IX=6: QI=2: TW=3: JY=20: JX=8
1280 FOR J=0 TO 5: N(J)=0: NEXT J
1290 REM  ============================
1300 REM  **** Plotting bits of screen.
1310 REM ============================
1320 PLOT 64,57,LV$+">="+N$(P0)
1330 PLOT 38,57,W$+N$(N(1))+N$(N(2))+N$(N(3))+N$(N(4))+N$(0)
1340 PLOT 0,57,Q$+N$(6)
1350 GOSUB 2910
1360 GOTO 470
1370 REM ========================================
1380 REM **** Data of the characters to be defined.
1390 REM ========================================
1400 DATA 56,60,16,252,186,185,56,40,43,252
1410 DATA 0,0,0,0,0,0,0,0,15,241,1
1420 DATA 0,0,0,0,0,0,0,0,31,224,0,0
1430 DATA 0,0,0,0,0,0,126,128,0,0,0
1440 DATA 0,0,0,0,0,0,126,1,0,0,0
1450 DATA 0,0,0,0,0,0,0,248,7,0,0
1460 DATA 0,0,0,0,0,0,0,0,224,159,128
1470 DATA 28,60,8,63,93,157,28,20,148,126
1480 DATA 0,56,60,17,126,184,184,60,100,70
1490 DATA 0,28,60,8,62,93,157,28,22,50
1500 DATA 0,254,56,56,56,24,152,152,112,0
1510 DATA 0,198,198,206,206,206,254,254,124,0
1520 DATA 130,198,254,214,214,214,214,198,198,0
1530 DATA 0,252,254,198,198,254,252,224,224,0
1540 DATA 0,254,124,56,56,56,56,124,254,0
1550 DATA 0,230,246,246,222,206,239,231,231,0
1560 DATA 0,62,126,112,96,110,102,126,60,0
```

```
1570 DATA 0,0,0,0,0,0,0,0,0,0
1580 DATA 0,60,102,102,126,126,102,102,36,0
1590 DATA 0,252,230,198,206,252,230,198,252,0
1600 DATA 0,254,124,56,56,56,56,124,254,0
1610 DATA 0,224,224,224,224,240,254,254,124,0
1620 DATA 0,254,124,56,56,56,56,124,254,0
1630 DATA 0,124,254,254,56,56,56,60,30,0
1640 DATA 0,102,102,62,14,142,206,254,124,0
1650 DATA 0,62,126,96,124,62,6,126,124,0
1660 DATA 0,60,126,254,224,224,254,126,60,0
1670 DATA 0,124,254,198,198,230,230,254,124,0
1680 DATA 0,120,204,204,252,248,236,230,230,0
1690 DATA 0,124,254,192,248,248,192,254,124,0
1700 DATA 0,0,14,138,255,72,40,24,8,0
1710 DATA 8,24,40,72,78,170,255,0,0,0
1720 DATA 16,24,20,18,114,85,255,0,0,0
1730 DATA 0,0,112,81,255,20,18,20,24,16
1740 DATA 0,124,254,202,146,146,230,254,124,0
1750 DATA 0,24,56,120,24,24,24,60,126,0
1760 DATA 0,60,102,66,6,28,48,124,254,0
1770 DATA 0,124,254,6,62,62,6,254,124,0
1780 DATA 0,28,56,112,228,228,254,126,4,0
1790 DATA 0,126,96,96,124,62,14,126,252,0
1800 DATA 0,120,252,192,252,254,198,198,124,0
1810 DATA 0,124,254,14,28,56,112,224,192,0
1820 DATA 0,124,198,130,124,124,130,198,124,0
1830 DATA 0,60,102,102,62,2,70,108,56,0
1840 DATA 0,224,224,224,224,240,254,126,60,0
1850 DATA 0,254,124,56,56,56,56,124,254,0
1860 DATA 0,198,198,198,238,238,238,124,56,0
1870 DATA 0,124,254,192,248,248,192,254,124,0
1880 DATA 0,124,252,192,248,124,12,252,120,0
1890 DATA 0,255,255,8,20,34,65,128,255,255
1900 DATA 5,10,18,21,42,40,42,21,10,5
1910 REM ============================================
1920 REM **** Checking whether the player has caught a
1930 REM **** bird.
1940 REM ============================================
1950 FORJ=1TO9
1960 IF JX<BX(J)-2 OR JX>BX(J)+2 OR JY<BY(J)-3 OR JY> BY(J)+3 THEN NEXT J: GOTO 840
1970 REM ============================================
1980 REM **** Since a bird has been caught, the  score
1990 REM **** will increase by 10, the dead bird  will
2000 REM **** be wiped out and given new  co-ordinates
2010 REM **** in the form of a new bird.
2020 REM ============================================
2030 SC=SC+10
2040 PLOT BX(J),BY(J)," "
2050 IF BQ(J)=1 THEN BX(J)=0
2060 IF BQ(J)=2 THEN BX(J)=80
2070 BY(J)=INT(RND(1)*16)+3
2080 N(4)=N(4)+1
2090 FOR J=4 TO 1 STEP -1
2100 IF N(J)>9 THEN N(J-1)=N(J-1)+1: N(J)=0
2110 NEXT J
2120 PLOT 38,57,W$+N$(N(1))+N$(N(2))+N$(N(3))+N$(N(4))+N$(0)
2130 GOTO 840
2140 REM =======================================
2150 REM **** Clearing, reprinting the screen and
2160 REM **** giving the variables new value  now
2170 REM **** the player has lost a life.
2180 REM =======================================
2190 IQ=1: XX=0.5: YY=3: X=8
2200 IX=6: QI=2: TW=3: JY=20: JX=8
2210 GOSUB 3000                  :REM ** GOSUB's to screen clearing.
2220 PO=PO-1
2230 PLOT 64,57,LV$+">="+N$(PO)
2240 REM =======================================
2250 REM **** Checking if the player has lost all
2260 REM **** his lives.
2270 REM =======================================
```

```
2280 IF PO<1 THEN GOTO 2360
2290 PLOT 0,57,Q$+N$(6)
2300 GOSUB 2910
2310 GOTO 260           :REM ** Returning to main loop.
2320 REM ========================================
2330 REM **** The player has lost all  his  lives
2340 REM **** and the game finishes.
2350 REM ========================================
2360 PLOT 31,30,"GAME OVER"
2370 FOR J=1 TO 1000: NEXT J
2380 REM ========================================
2390 REM **** Seeing if the player is in the  high
2400 REM **** score board.
2410 REM ========================================
2420 IF SC<HS(10) THEN GOTO 2600 :REM ** Checking if the player's score is
2430                             :REM ** less than the lowest score.
2440                             :REM ** If it is then GOTO (HIGH SCORE).
2450 FOR J=10 TO 0 STEP -1
2460 IF SC>HS(J) THEN HS(J+1)=HS(J): HS$(J+1)=HS$(J)
2470 IF SC<HS(J) THEN J=J+1: GOTO2490
2480 NEXT J
2490 HS(J)=SC
2500                             :REM ** Finds where the player is in the
2510                             :REM ** high score and inserts the score.
2520 REM ========================================
2530 REM **** Placing the players score and name into
2540 REM **** the high score board.
2550 REM ========================================
2560 PLOT 0,27,"Congratulations you have a high score..."
2570 FOR U=1 TO 12: PUT 11: NEXT U
2580 INPUT "Please input your name";HS$(J)
2590 FOR J=1 TO 11: PUT 10: NEXT J
2600 GOSUB 3000                  :REM ** GOSUB's to screen clearing.
2610 REM ================================
2620 REM **** Plotting the high score board.
2630 REM ================================
2640 PLOT 20,54,"Nicksoft's FLIP-FLOP"
2650 PLOT 20,51,"===================="
2660 PLOT 28,45,"Hall of fame"
2670 FOR J=1 TO 10
2680 PLOT 0,42-(J*3),HS$(J)+"....................................."
2690 PLOT 70,42-(J*3),STR$(HS(J))+"  "
2700 NEXT J
2710 PLOT 14,0,"Press spacebar to continue"
2720 PLOT 22,-3,"Press ESC to stop."
2730 G=GET():IF G=27 THEN PUT 31:END
2740 IF G<>32 THEN GOTO 2730
2750 GOSUB 3000                  :REM ** GOSUB's to screen clearing.
2760 REM ========================================
2770 REM **** Initialising variables and starting the
2780 REM **** game again for another go.
2790 REM ========================================
2800 IQ=1: XX=0.5: YY=3: X=8: PO=3
2810 IX=6: QI=2: TW=3: JY=20: JX=8: SC=0
2820 PLOT 64,57,LV$+">="+N$(PO)
2830 FOR J=1 TO 4: N(J)=0: NEXT J
2840 PLOT 38,57,W$+N$(N(1))+N$(N(2))+N$(N(3))+N$(N(4))+N$(0)
2850 PLOT 0,57,Q$+N$(6)
2860 GOSUB 2910
2870 GOTO 260           :REM ** Going to main loop.
2880 REM ========================================
2890 REM **** Plotting out the Iron Girders and walls.
2900 REM ========================================
2910 FOR J=1 TO 10
2920 I=INT(RND(1)*80)+1: O=((INT(RND(1)*14)+1)*3)+9: U=INT(RND(1)*10)+1
2930                     :REM ** Randomly selecting the place and length of the
2940                     :REM ** girders.
2950 FOR TU=1 TO U
2960 PLOT I+(TU*2),O,CHR$(178)
2970 NEXT TU,J
2980                     :REM ** Plotting the girders.
2990 RETURN
```

```
3000 REM ========================
3010 REM **** Cleaning the screen.
3020 REM ========================
3030 FOR J=54 TO -3 STEP -3
3040 PLOT 0,J,"                                        "
3050 NEXT J
3060 RETURN
3070 REM ================================================
3080 REM **** Making the man bounce off the force fields
3090 REM **** at the sides.
3100 REM ================================================
3110 PLOT JX+XX,JY-3,CHR$(179)
3120 PLOT JX+XX,JY,CHR$(179)
3130 IF JY<54 THEN PLOT JX+XX,JY+3,CHR$(179)
3140                    :REM ** Plotting the shock of the force field.
3150 PLOT JX+XX,JY-3," ": PLOT JX+XX,JY," ": IF JY<54 THEN PLOT JX+XX,JY+3," "
3160                    :REM ** Unplotting the shock of the force field.
3170 IF JX<2 THEN XX=ABS(XX)
3180 IF JX>76 THEN XX=-ABS(XX)
3190 JX=JX+XX           :REM ** Changing the direction of the man after hitting
3200                    :REM ** a force field.
3210 RETURN
```

Douai School,
Upper Woolhampton,
Reading,
RG7 5TH.

# Programming with WORDSTAR

*Neil Howie*

One of the principle reasons for the rise in popularity of SBAS as a dialect of BASIC is its extensive built-in facilities for structuring programs in a PASCAL-like way.

Simple programs, and with careful planning, quite long and complex ones, can be written in SBAS without any need to use the unfashionable keywords GOTO or GOSUB and you don't even have to use THEN if you don't want to. This of course assumes that you can manage without ON BREAK or ON ERROR statements, and you put your own end of file markers in files, never leaving empty files cluttering up your disc whose presence needs detecting with ON EOF.

One consequence of this is that it is now possible to write these programs without line numbers, which of course makes SBAS even less like BASIC, because it accepts and LOADS them just the same! Once there, line numbers can be suppressed with the console command DISPLAY FALSE, one of several useful options which can, and probably should, be preset using the configuration program supplied with SBAS.

Most teachers of computing with enough equipment to give their pupils a reasonable amount of access will long ago have discovered that most of their charges will, given half a chance, develop their programs a few lines at a time, with much trial and error, at the keyboard. In fact, if we are honest, most of us probably do too, much of the time! As a means of combatting this, we conducted a trial at Oakwood in 1984/85 in which one of our computer studies groups was asked to prepare the first draft of all their programs using a non-formatting version of TXED which, to emphasise its purpose, we called PREP[1]. We found, as hoped, that this approach did indeed tend to encourage pupils to think more carefully about their programs in advance; however, the time involved in reloading PREP, and the fact that we only had RML BASIC at the time, meant that program editing continued to be done in the oldfashioned, suck-it-and-see, way.

[1]To rid yourself of the formatter and its commands use O*EX$$ in command mode with no file in memory and type SAVE nn PREP.COM where nn is the number given in your release note.

So it was that this school year, taking courage in both hands, we decided to introduce all our new classes to programming through WORDSTAR. The hesitation over making any use of WORDSTAR at all since it arrived with our network bundle was largely due to an ingrained familiarity with TXED and the reluctance to learn a completely new set of control commands: indeed the frontispeice of the WORDSTAR manual with its *two*-character codes seems thoroughly forbidding. It also doesn't help that the first half of the manual appears to be telling you how clever WORDSTAR is without telling you much about how to use it, yet like any software, one has to sit down and use it to find out enough about it to appreciate it.

What, then, makes WORDSTAR such an attractive proposition?

Primarily, the fact that it can be, in its own simple way, a complete development system. Not quite in the style of Microsoft Windows on the Nimbus, but nevertheless allowing you to process and make use of the files it creates in a somewhat more friendly way than using the system commands of CP/NOS. The feature that makes this possible is a number of single key commands which appear in a 'No file menu' at the top of the screen when you first load WORDSTAR. With the aid of these you can load or create a file, delete, copy or rename a file, even get a directory listing or change your logged on drive, but probably most useful, is the command R for run.

With this you can *temporarily* suspend execution of WORDSTAR and start running any other program you like. For example, you could use STAT to find the amount of space on your drive, and when it has responded, you are invited to press any key to return to WORDSTAR. If your drive hasn't enough space for the file you wish to create, you can then change drives (but only until you leave WORDSTAR). However, more relevant to our purpose, you can run SBAS (or any other program for which you have prepared a source file- does anyone else but me use ALGOL?) and forget all about WORDSTAR for now, loading your program as usual and running it.

Now comes the clever bit. Having written your program, without line numbers, the DISPLAY FALSE flag has caused it to be loaded with lines numbered 1,2,3,. . ., none of which appear on the screen! What to do about the inevitable errors? Straightforward syntax errors and certain others are O.K. Rather like RML version 6, typing EDIT return will then display the offending line on the screen in edit mode so that you can put it right. But 'Procedure 'FRED' missing at line 8', can cause a bit of a headache. You can type things like RENUMBER and DISPLAY TRUE (if you know about them), but that, like using GOTO, is cheating, isn't it!

The solution is simple; typing control C or BYE will return you to the 'Any key for WORDSTAR' prompt, so that you can then go back immediately, edit the program and hopefully get it right before trying again. *Even cleverer*, at the stage when you are asked the name of the file to edit, WORDSTAR remembers it, and will reveal it by typing ^Y or repeating the 480Z right arrow key! (This can be useful for the request to print, too)

What then about the practicalities of using WORDSTAR? If you are already used to it, then you can skip the rest, but there may still be some bits in the section on patching to interest you. This is meant for those who, like myself earlier this year, have not yet dared to try it out seriously.

**How to use it**

Let's assume that you have done all the necessary things to log onto a drive containing a copy of the standard RML version called WS.COM and the two overlay files WS*.OVR.

Type WS after the prompt; there will be a pause whilst WORDSTAR is loaded, then a sign-on message and another long pause before the <<<NO FILE MENU>>> appears at the top of the screen. This is the menu that tells you which single letter keys to press for all those useful things like renaming files and getting a directory and so on. It also tells you that you have two choices for opening a file: document mode (key D) for nicely formatted letters and things, and nondocument mode (key N) which is the one we want for writing our programs. So, what are you waiting for? Type N, then! After the prompt, type the name of any old and useless program for experimenting with.

There will be another delay, partly due to WORDSTAR looking to see if there is already a file of that name to edit, and partly just to keep you waiting[2] and to make sure you've read any messages that might appear.

Now you are nearly ready to edit your program. Firstly though, let's take a look at the top of the screen. This is taken up with the <<<MAIN MENU>>>. Much of this is unnecessary on a 480Z, because most of the single control key commands (indicated by the ^ symbol) are implemented in a fairly obvious way by the editing keypad. The best thing is to play about with these keys until you get the hang of it. Watch one particular major difference from TXED- shift F1 deletes the whole line, not just one word! Of the other commands, one of the more useful is ^N which is the key to press if you want to insert extra lines of program code where the cursor is.

At the right hand side of the menu is a table of the other menus you can call up. Try pressing ^K, then waiting; the <<<BLOCK MENU>>> will appear. This will list a set of single key commands for doing various thigs with blocks of text, like moving them around. Remember also that your whole program is a block, and that moving around extends to saving your program on disc, which you can do occasionally as you go along with the block menu's S command for peace of mind, or when you have finished, with D (for done).

[2]One of the things which takes some getting used to is that if you know what the next command or word is that you wish to type, you can cut short some of these pauses, but beware – do it at the wrong time and you could find an unexpected prompt at the top of the screen from which the safest way out is to hit the ESCape key.

Just a minute, though, what about those two key commands we mentioned earlier? Well, this is the trick: if you know what the appropriate command from another menu is, why wait for that menu to appear? For example, to abandon (quit) the file you are playing with, type ^KQ. It doesn't matter if you keep the control key down when you do it. Typing the Q straight after puts the command straight into effect, and saves a lot of time as well.

There is also a print menu, ^P, which puts special codes into your text to make an Epson printer underline, do bold type and other things, an onscreen menu, for certain visible formatting jobs (^OC is useful for centering text on the line) and a 'quick' menu, ^Q, of miscellaneous things the most useful of which are probably moving the cursor to the start or end of a line (^QS and ^QD) and finding that elusive variable with ^QF. In fact I have found ^QS and ^QD are so useful that I have reconfigured my F3 and F4 keys to execute them. Details are given later.

Back to the job in hand. Before proceding, if you have been experimenting whilst reading these notes like I asked you to, type ^KQ to return to the <<<NO FILE MENU>>>, then open a new, nondocument file called MYPROG.BAS or something. During the inevitable delay, do *not* type a return to wake WorsStar up – it will go into your file without you realising it and when SBAS finds a file starting with a carriage return, it seems to think you want your program numbered from 1537 or something ridiculous like that, which can make error messages a little confusing, to say the least. Try delete for safety.

Armed with this preamble, you can now, with great confidence, type ^OV. Nothing will seem to happen, except for a vague flicker in the top left of the screen as the command registers, but *don't do it again!* This little gem is essential if you intent to use ^I for tabbing your line indents. If you don't do it, you will find your SBAS program full of invisible syntax errors! If this does happen, help is at hand in yet another of my little footnotes[3].

So, off you go. Try to remember your favourite iterative solution of quadratic equations program, or file merging program, and type it in. When you think you've done, ^KD to save the program, R to bring up the COMMAND prompt – reply SBAS (or whatever you call it) – then LOADGO '. . .'. To edit your errors (what do you mean, there are none?) type ^C (you are sure) then NN (yes two) when it says "Hit any key to return to WORDSTAR" and follow the NAME OF FILE TO EDIT? with a right arrow key, repeat and return. There's your program ready for amendment.

[3]Type EDIT, and the offending symbol, a shy little sideways triangle representing an ASCII 9 (^I to you) will reveal itself ready to yield to your delete key. Typing ^OV will ensure all tabs are expanded with 5 spaces and the problem will not arise. If you do have a program with a lot of these nasties in, it is worthwhile using PIP to sort them out. From WORDSTAR <<<NO FILE MENU>>>, select R, then after the prompt COMMAND? enter PIP newfilename:=oldfilename[T5], then any key to return to WORDSTAR when asked.

When you've had enough, ^KX will take you back to the operating system *and* save your file from the <<<MAIN MENU>>> or just X will take you out from the <<<NO FILE MENU>>>. One reason for going back to the <<<NO FILE MENU>>> before exiting, is that you might want a listing of your program as typed. P is the command which will do it, although ^KP works as well when you have a file in memory (save the present version with ^KS first). Note that you can *only* print a file that has been saved on disc, not the image in memory which you are presently editing. This is a safety device to make sure you always have a copy saved, and a check will be made to see that the file does exist. To be able to initiate printing, WORDSTAR prompts for the filename (try ^Y first) then begins a series of questions. *do not* answer any of these questions! Simply press ESCape and printing will begin.

Well, that's all for this bit. Over the page are details of how and what to patch to create a version of WORDSTAR which is geared a bit more towards program preparation than word processing.

**Patching WORDSTAR**

Many articles have appeared in past issues both on the subject of patching in general (i.e. changing bits of a program to make it do what you want it to do) and on patching WORDSTAR in particular. However, there may be new subscribers for whom the finer details which follow will be useful, so I crave the indulgence of the expert.

There are three main ways of patching machine code programs like WORDSTAR. One is to use another program specially geared to do the job. MICROPRO give you such a program, but if you use it, you then have all sorts of other problems because RML supply two more patches of their own which have to be incorporated too. The others involve loading the program into memory first (using the utility program DDT) then either poking around in memory with DDT or the Front Panel, or else writing a piece of assembly code, assembling it and merging it.

I am going to outline each of these, then give a list of those changes which I consider desirable. Where 'd:' appears, this will refer to the drive on which the program or file following it appears. All numbers, except that in the SAVE command, are in hexadecimal.

Firstly, log into the drive on which your copy of WORDSTAR is saved, then to get it into memory from the logged on drive type:

```
d:DDT WS.COM
```

The computer responds in the usual user friendly way with

```
NEXT  PC
4000 0100
-
```

If the numbers are outside this range and you are convinced no-one else has been playing about with WS, you may have a different version from the common 3.00 one; RML may be able to help! These numbers tell you where the machine code starts and where it finishes.

Now to look at the memory and to change it. Typing ^F at this point takes you into front panel. For the uninitiated, this is two bewildering arrays of numbers, but we wizards immediately recognise two clear arrows almost in the middle of the bottom block between which is a memory address followed by the memory contents. The address is probably 0100 and the contents C3 if you started from cold. Typing M and then a hex number up to 4 digits will move the memory display so that the address you typed is between the pointers (note the > symbol after the M which tells you when to type). Typing a number without the M will put its last two digits after the address, changing the memory contents. So suppose a patch suggests setting the contents of address 02D2 to 0, you would type

```
M2D2 return Ø return
```

When all changes have been made in this way, you can return to the main display by typing K.

One disadvantage of using the front panel is that if you make a mistake, you may not know what the contents of an address were before you changed them, so correction is difficult unless you record everything you do. The alternative is to stick with DDT. The command to change a memory location is S followed immediately, no spaces, by the address. The old contents are shown alongside and the cursor requests you to enter the new contents.

So the same change described above would look like

```
-s2d2

02D2  40  00

02D3  00
```

and if you don't want to change anything else near this address, type a full stop to get back to the '-' prompt, otherwise you can skip through with returns.

There is one other essential change to be made if you are going to save this version with a different name, rather than renaming the old version. The new name, no more than 8 characters long, must be put at location 03E7 hex. Failure to do this will mean you wil not be able to run other software from within WORDSTAR. The job is best done with Front Panel. (Type ^F if in DDT.) On a 480Z, type T to turn ASCII codes in the memory display into characters, M3E7 to get to the location, then an opening double quote followed by the new name in upper case (no closing quote) and return. Type K to return the machine to its previous state. On a 380Z you will have to enter the name using ASCII codes, one at a time.

When all changes have been made, you leave DDT by typing g0. (That means GOTO 0- the CP/M restart address. If you really hate GOTOs enough, you could try ^C). Then think of a name for your new version, which must end with .COM and type the magic

```
SAVE 63 newname.COM
```

The third way does not involve loading DDT at this stage. You have to create a text file (using WORDSTAR of course!) of couplets like this:

```
org     2D2 H
defb    00 H

org     next address
defb    new contents
```

In addition, you will need a couplet like

```
org     03E7 H
defm    'WSNEW'  or whatever your new name is  (in  single quotes)
```

to allow WSNEW to load other software like SBAS.

Now save this file on disc with a name like WSPATCH.ZSM – the ZSM is obligatory. Make sure you have ZASM.COM available (the RML assembler) and type

```
d:ZASM WSPATCH
```

which creates a file called WSPATCH.HEX whose name you should find in your directory. To merge this with your version of WORDSTAR, engage in the following dialogue (type the underlined text):

```
d:DDT WS.COM

NEXT  PC
4000 0100
-IWSPATCH.HEX
-R

NEXT  PC
4000 nnnn          (it doesn't matter what nnnn is)
-GO

d>SAVE#63#newname.COM
d>
```

You are now ready to try it out by typing its new name. Good luck!

### Locations to patch and why

All addresses and entries are in hexadecimal; if preparing a ZSM file, don't forget to add H afterwards and a 0 in front of any hexadecimal number beginning with a letter A to F, e.g. 0FF H . The order is address, old value, suggested new value, explanation.

| | |
|---|---|
| 02D2 40 00 | cuts out some useless messages and reduces some of the horrible delays |
| 0363 FF 00 | prevents great long directory listings cluttering up your screen |
| 0392 00 FF | makes sure that if you start up with WS MYPROG.BAS your file is opened in *non-document* mode |

The following set of modifications help you get more on your printer paper and cut out page numbers:

| | |
|---|---|
| 037E 08 04 | makes left margin on *printout* narrower |
| 0380 40 48 | makes line 8 characters longer. Use these two together |
| 036B 03 02 | these two reduce |
| 036C 18 10 | the heading margin |

| | |
|---|---|
| 0373 02 01<br>0374 10 08 | these two reduce the bottom margin – useful particularly for FX printers |
| 03D3 00 FF | this suppresses page numbers |

These you will find worth changing as the year goes by:

| | |
|---|---|
| 0360 03 02 | removes the cursor and display movement menu which gets in the way of text and slows down menu changes. Values of 01 and 00 reduce help and speed things further. Help is restored with ^JH3 if required. |
| 0361 FF 00 | This gives a message about using ^JH3. |

I find the following redefinitions of the F3 and F4 keys more useful:

| | |
|---|---|
| 3FC5 04 27 | cursor to beginning of current line |
| 3FC4 33 09 | cursor to end of current line |

This next one is necessary if you wish to load WORDSTAR's overlay files (with names beginning WS and ending .OVR) from your silicon disc or any other drive:

| | |
|---|---|
| 02DC 01 nn | nn is the drive with the files on 01=A, 02=B, . . . . . 0D=M |

As already mentioned, if you are calling your new version anything but WS:

| | |
|---|---|
| 03E7 and following bytes (up to 8 altogether) | The name of your new version |

Further details will be found in the *WORDSTAR User Guide* which comes with your disc, in the *Configuration Guide* which has to be bought from MICROPRO and, *free from* RML, in *Technical Information Bulletins 20* (Reprinted in the Read Using 15) and 24 and various release notes.

**Disclaimer!!!**

I have tried to ensure the accuracy of the above. My WORDSTARs work but I can't rule out transcription errors. If you haven't kept a backup copy of your original . . . . .

Oakwood Park Grammar School,
Maidstone,
Kent, ME16 8AH

# GCSE
# CONTROL INTERFACE

## At last! An interface that's easy to use.

Touch the inputs and get immediate results.

Simple commands.

All sockets are 4mm.

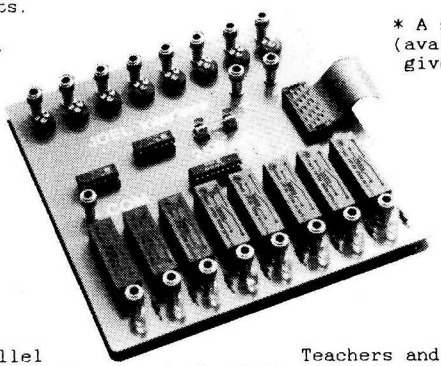
8 bit output LED display

For 380/480Z

On board power.

Plugs into parallel I/O port via 1.5m cable.

* Ideal for use with Unilab alpha kits



* A simple addition (available from us) gives full control of your Leggo buggies.

*Integrated traffic light module

Developed by Teachers and used by pupils in North Tyneside for GCSE project work

SUPPLIED WITH ILLUSTRATED USER GUIDE, EXAMPLE PROGRAMS AND CIRCUIT SUGGESTIONS

ALSO LIGHT SENSOR AND BUZZER UNIT TO GET YOU STARTED

General Specifications:
:8 BIT commoned relay switched outputs rated 500mA.30V
:8 BIT touch sensitive comparator inputs
:On board 5v 250mA fused supply

Price: £83.50 inclusive

Allow 28 days for delivery

JOEL ELECTRONICS

39 Manor House Road,
Jesmond,
Newcastle Upon Tyne
NE2 2LY (Tel. 091 2811489)

# Combinations of N things taken K at a time.

*by Dr. A. Hall*

Look in any text on a structured language and you are likely to find an illustrative example of a routine for the recursive generation of the Factorial of the first N positive integers. Look for a routine to generate all permutations of the first N positive integers or a routine to produce combinations of the first N positive integers taken K at a time and you will be lucky to find one. However, a little research will reveal that a lot of creative work has been done in the field of Combinatorial Analysis. For example, the publication *Combinatorial Algorithms* by Albert Nijenhuis and Herbert S. Wilf published by Academic Press, page 22, gives a very concise algorithm as follows :

**Next K-subset of an N-set in lexicographic order.**

```
(A) (First entry) a(j) := j(j=1,k); Exit.

(B) (All later entries) h := min (j   s.t. a(k+1-j) <> n+1-j );

                        m(1) := a(k+1-h); a(k+j-h) := m(1)+j (j=1,h);

                        If a(1)=n-k+1, final exit;

                                Exit.
```

The first is in BASIC which exposes the process. At the other extreme a Z80 Assembly level program allows another interpretation. In this latter approach the start of the vector of K locations is placed on a page boundary and N = 127 is taken as the upper possible limit for the total number of consecutive integers in the set. An algorithm for the generation of the Next Permutation of the first N Positive Integers can be found in Nijenhuis and Wilf chapter 7.

**RML BASIC Program**

```
100 REM>            * This routine generates and      *
110 REM>            * prints out all the combinations *
120 REM>            * of the first N positive         *
130 REM>            * integers taken K at a time.     *
140 REM             * ******************************** *
150 REM>--- Initialise.
160 INPUT "Input K and N"; K,N
170 IF (K<=0)OR(N<=0)OR(K>N) THEN 160
180 DIM A(K)
190 REM*******************************
200 REM>--- Set up the initial vector of K
210 REM>--- elements A(J)=J;J=1,..K.
```

```
220 M=0
230 H=K
240 GOSUB 490
250 REM*******************************
260 REM>--- Test for the condition
270 REM>--- A(1)=N-K+1;if so final exit
280 REM>--- else continue...
290 IF A(1)<>N-K+1 THEN GOSUB 370  ELSE END
300 GOTO 290
310 REM***********
320 REM>--- Search for the least value
330 REM>--- of H such that the vector
340 REM>--- element A(K-H+1)<>(N-H+1)
350 REM>--- if this exists set M=A(K-H+1)
360 REM>--- and exit from the search.
370 FOR H=1 TO K
380 M=A(K-H+1):P=(N-H+1)
390 IF M<>P THEN GOSUB 490:H=K
400 NEXT H
410 RETURN
420 REM***********
430 REM>--- Use the values of H and M
440 REM>--- found by the least value search
450 REM>--- to generate H new elements in
460 REM>--- the K-vector and then call the
470 REM>--- routine to display the new
480 REM>--- vector in full.
490 FOR J=1 TO H
500 A(K+J-H)=M+J
510 NEXT J
520 GOSUB 560
530 RETURN
540 REM************
550 REM>--- Display the K-vector.
560 FOR I=1 TO K
570 ? A(I);
580 NEXT I
590 ?
600 RETURN
610 REM*************
```

RML Z80 Assembly Level Program.

```
                ;RML EMTs used by the machine dependent routine XXXX:
                ;
0000 =          CONTC   EQU     00      ;Return to command level.
0001 =          OUTC    EQU     01      ;Output byte in regd.A to screen.
0017 =          MSG     EQU     017H    ;Output message at (HL) to screen.
0024 =          SAVEA   EQU     024H    ;Save regs.AF,BC,DE and HL.
00FF =          TERM    EQU     0FFH    ;Message terminator.
                ;COMBINATORIAL parameters:
                ;
0002 =          PAGE    EQU     02      ;For example.
0004 =          K       EQU     04      ;For example.
0006 =          N       EQU     06      ;For example.
                ;
                ;Working constants:
                ;
0020 =          SPACE   EQU     020H
000D =          CR      EQU     0DH
000A =          LF      EQU     0AH
0100                    ORG     0100H   ;For example.
                ;
0100 CD1501     START:          CALL INTK
0103 E5         MOREK:          PUSH HL
0104 CD6001                     CALL XXXX
                ;
                ;  The CALL to routine XXXX is user dependent
                ; and would be employed to display or use the existing
                ; K-vector in some way before the next vector
                ; is generated from it on the next cycle
                ; of the main program.
                ;
```

```
0107 CD4801                     CALL MIN
010A E1                         POP HL
010B CD2E01                     CALL EXTST
010E 20F3                       JR NZ,MOREK
0110 CD6001                     CALL XXXX
                        ;.................
0113 F700                       EMT CONTC
                ;Machine dependent Exit to Command level.
                        ;*******************
                ;INITIALISE the variables and K-vector.

0115 DD215D01 INTK:             LD IX,PARBLK
0119 DD6600                     LD H,(IX+0)
011C DD6E01                     LD L,(IX+1)
011F DD4E02                     LD C,(IX+2)
                ;
0122 79                         LD A,C
0123 95                         SUB A,L
0124 3C                         INC A
0125 08                         EX AF,AF'
0126 45                         LD B,L
0127 1600                       LD D,00
0129 72                         LD(HL),D
012A CD3801                     CALL PUTK
012D C9                         RET
                        ;********************
                ;EXIT TEST.

012E 54         EXTST:          LD D,H
012F 1E01                       LD E,01
0131 EB                         EX DE,HL
0132 08                         EX AF,AF'
0133 BE                         CP A,(HL)
0134 C8                         RET Z
0135 EB                         EX DE,HL
0136 08                         EX AF,AF'
0137 C9                         RET
                        ;********************
                ;MAKE NEW K-VECTOR.

0138 F5         PUTK:           PUSH AF
0139 C5                         PUSH BC
013A E5                         PUSH HL
013B 5E                         LD E,(HL)
013C 7A         LOOP1:          LD A,D
013D 80                         ADD A,B
013E 6F                         LD L,A
013F 7B                         LD A,E
0140 80                         ADD A,B
0141 77                         LD (HL),A
0142 10F8                       DJNZ LOOP1
0144 E1                         POP HL
0145 C1                         POP BC
0146 F1                         POP AF
0147 C9                         RET
                        ;********************
                ;LOOK FOR LEAST value of H such that
                ; A(K-H+1) <> (N-H+1)

0148 0601       MIN:            LD B,01
014A 7D                         LD A,L
014B 3D                         DEC A
014C 57                         LD D,A
014D 3C         TOP:            INC A
014E 6F                         LD L,A
014F 79                         LD A,C
0150 90                         SUB A,B
0151 3C                         INC A
0152 BE                         CP A,(HL)
0153 C43801                     CALL NZ,PUTK
0156 C0                         RET NZ
```

```
0157 04                     INC B
0158 15                     DEC D
0159 F8                     RET M
015A 7A                     LD A,D
015B 18F0                   JR TOP
                    ;**********************
                ;PARAMETER BLOCK of the values supplied
                ; to the combinatorial routine by the user.
                ; The VALIDATION of the data is assumed
                ; to have already taken place!
015D 02         PARBLK: DEFB    PAGE    ;PAGE boundary.
015E 04                 DEFB    K       ;K s.t. 0 < K <=N
015F 06                 DEFB    N       ;N s.t. K =< N < 256
                ;*****************************************
                ;XXXX demonstration routine using RML EMTs.
                ;This routine  will display successive
                ;K-vectors for any case where N<10.
                ;*****************************************
                ;
0160 F724       XXXX:           EMT SAVEA
0162 DD6600                     LD H,(IX+0)
0165 2E01                       LD L,01
0167 DD4601                     LD B,(IX+1)
016A 7E         LOOP2:          LD A,(HL)
016B E60F                       AND 0FH
016D FE0A                       CP A,0AH
016F F48701                     CALL P,ERR
0172 F0                         RET P
0173 F630                       OR A,030H
0175 F701                       EMT OUTC
0177 3E20                       LD A,SPACE
0179 F701                       EMT OUTC
017B 23                         INC HL
017C 10EC                       DJNZ LOOP2
017E 3E0D                       LD A,CR
0180 F701                       EMT OUTC
0182 3E0A                       LD A,LF
0184 F701                       EMT OUTC
0186 C9                         RET
                    ;******************
0187 3E0D       ERR:            LD A,CR
0189 F701                       EMT OUTC
018B 219201                     LD HL,ERRMSG
018E F717                       EMT MSG
0190 F700                       EMT CONTC
                    ;******************
0192 4572726F ERRMSG: DEFM      'Error: DISPLAY IS '
01A4 52455354         DEFM      'RESTRICTED TO N<10 '
01B7 FF               DEFB      TERM
0100                  END       0100H

0000 CONTC   000D CR      0187 ERR     0192 ERRMSG  012E EXTST
0115 INTK    0004 K       000A LF      013C LOOP1   016A LOOP2
0148 MIN     0103 MOREK   0017 MSG     0006 N       0001 OUTC
0002 PAGE    015D PARBLK  0138 PUTK    0024 SAVEA   0020 SPACE
0100 START   00FF TERM    014D TOP     0160 XXXX

No errors
```

Demonstration Run.

```
With K=4 and  N=6 :

                1  2  3  4
                1  2  3  5
                1  2  3  6
                1  2  4  5
                1  2  4  6
                1  2  5  6
                1  3  4  5
                1  3  4  6
                1  3  5  6
                1  4  5  6
                2  3  4  5
                2  3  4  6
                2  3  5  6
                2  4  5  6
                3  4  5  6


 ( 6 !/4 !)/ 2! = 15 K-vectors.
```

Computing Dept., The King's School, Tynemouth, TYNE and WEAR. NE30 4RF

# Feeling at home with your Nimbus

*by Ken Maynard*

*Sub-directories*, *Path Names* and *Batch Files* are expressions that anyone who has used a Nimbus for a while will drop into the conversation without a moment's thought. They might be called 'second order concepts'. A newcomer to computing would benefit greatly from being able to use them but they are one step removed from some of the absolutely essential ideas that have to be grasped first. Someone transferring from an earlier machine, however, should have no problem with the principles involved. These notes are an attempt to speed their adoption by anyone who has delayed studying the relevant pages of the Owners Handbook and the MS-DOS guide.

**Configuring the Screen**

There are many features of the Nimbus screen display that are under the control of the user and, if desired, can be set up automatically on switching on. The most important of these is the facility to cause the 'prompt' to display the current 'path' or 'directory' instead of just 'A>' but the colours of the text and the background, and the size and style of the characters can also be modified if desired. It is possible to set up special 'paths' to enable subsequent easier use of the system. It really is one of the injustices of this life that only when one is experienced in using a complicated system can one exploit the simpler methods. The novice, because of his ignorance, must do things the long way!

**Directories**

One of the most useful features of MS-DOS, the Nimbus operating system, is its ability to support hierarchical directories. This does not immediately appeal to most newcomers because they do not realise the convenience this can bestow, nor is it explained in novice terms in the documentation. Also, to the user who investigates only partially it may appear to involve a great deal of additional and unnecessary typing. However, MS-DOS also provides elegant shortcuts that make the whole system a pleasure to use, especially after CP/M.

Hierarchical directories simply means directories or lists of files nested within other lists. Groups of related programs may be given appropriate collective names and these names too may be given an overall title. For example, if I write lots of letters to Mr Smith, Mr Brown, etc. I might create one directory called *Smith* and another called Brown Individual letters within each may be given titles like *Holiday France* etc.; whatever is their principal subject matter. I can then get at any item I want by using an extended name like *Smith\Holiday*. If, on my disk I also have other items like *Articles Reviews* and so on I would create a directory called *Letters* with *Smith Brown* etc. as sub-directories within that. The full name for my letter would then be *Letters\Smith\Holiday* This is a little lengthy but, users of lesser operating systems will appreciate, it is nice and easy to identify any file. On a winchester disk the use of such a scheme is clearly essential. It ought to be said that the actual creation of such directories is done the opposite way round to the above description.

Three principal directory commands exist.

MD Make Directory
CD Change Directory
RD Remove Directory

together with two special symbols \ and . .

The steps required to construct the above hierarchy would be, on first running the system, type:

```
MD LETTERS
```

and, if no other directory of the same name exists, that will be done. Type DIR if you wish, to check that it is now on the list.

Now 'change to' that directory by typing

```
CD LETTERS
```

Type DIR again and very little will appear, just two entries (dot & double dot) because this directory is empty. It is a pity that there is normally no indication on the screen that one is 'within' a particular sub-directory. Later it will be seen how this may be remedied.

Now type MD SMITH to make a directory within the LETTERS directory and, finally, type CD SMITH to get into the sub-sub-directory SMITH.

It is a good idea to picture a set of files stored on a disk within various directories rather like a family tree. The uppermost layer of this system, the normal entry level, is called the ROOT directory. To get back to that level type CD\ To 'rise' just one level type CD. . (ie double dot). There is a limit to the number of names in the root directory, just over 110, but no limit to the number within a sub-directory so sub-directories are vital if a large number of small files is to be stored.

## Using Directories

There are only two snags with directories. One is that some software does not accept names like SMITH\HOLIDAY as being a valid filename, but there are some easy ways round this. The other is that there is no standard utility supplied (other than BACKUP) to copy directory names complete with their contents, although such things do exist.

RM BASIC does, of course, accept extended filenames so that, in BASIC one could type:

```
SAVE "A:\LETTERS\SMITH\NEWPROG"
```

and the program NEWPROG.BAS will be stored within the sub-sub-directory SMITH.

Alternatively, and probably best, on entering the system type:

```
CD LETTERS\SMITH
```

and then every reference that does not specify some other directory will be taken to refer to LETTERS\SMITH. This includes commands like DIR, COPY, DEL, etc. However, to load BASIC, which we will presume is in the ROOT directory, one must type \BASIC rather than just BASIC while to save a program now simply needs:

```
SAVE "NEWPROG".
```

Because some software does not like such path names this second method is probably best used for all circumstances, although there is another, even neater but marginally more subtle method. Some unused disk drive letter, like 's', can be defined to stand for the longer LETTERS\SMITH and then SAVE "S:NEWPROG" will accomplish the same end. The command for this is

```
SUBST S: A:\LETTERS\SMITH
```

provided the program SUBST is on the appropriate disk, and assuming the directories referenced are on disk A.

Newcomers should appreciate that the use of sub-directories takes a little getting used to but it is so convenient eventually that it is worth experimenting as soon as possible. The WELCOME disk contains a sub-directory called WELCOME that can be investigated to get used to the idea.

## Batch Files

A batch file is a list of commands that one could type into the machine in order to make it do something, like load BASIC and then load the program "FRED" or change the text size to 40 column mode and change its colour to red or change directory to LETTERS\SMITH. The list of commands is typed in a word processor and then saved with a suitable name, like RED.BAT. Then, when the name RED is typed the computer behaves exactly as if the contents of the file had been typed one after the other. It is a facility that allows a single word to cause a whole sequence of actions that would be tedious to type very often. In practice, anyone doing considerable work on the machine is likely to build dozens of little 'batch' files each, effectively, an abbreviation for a sequence of frequently required tasks. A further feature is that one such file of commands can be selected to be run

automatically when the machine is switched on or reset. It is a particularly convenient way for a system manager to 'protect' novices from needing to learn many of the complexities of the computer before they can even use it.

Naturally, the first thing required before a batch file is created is a knowledge of what it is required to do. The usual tasks are to set up the 'environment' ( ie choose the desired colours, etc.) or to run a series of programs or, of course, both.

Suppose one wished to make the normal 'A>' prompt give a little more information and to run PAINTSPA that is in a sub-directory called 'PAINT'.

The batch file would say:

```
echo off        ;do no print the following commands on the screen
prompt $p       ;show the 'path name' in the prompt
cd paint        ;change directory to 'PAINT'
paintspa        ;load and run PAINTSPA
cd\             ;on leaving PAINTSPA revert to the ROOT directory
echo on         ;display commands again in normal way
```

Note, only the commands, not the remarks given above should be typed, although remarks can be incorporated; see the documentation. Do not type a RETURN after the last command or the prompt will display twice on exit, rather untidily. This file should be saved called, for example, JILL.BAT. The user, Jill presumably, has only then to type JILL for the above sequence to be obeyed. Two essentials are that the file is a 'text' file rather than a document file, and that its name ends with .BAT.

A text file can be created most easily in RML's text editor called STEED but Oxfordshire's WRITE (use esc-T,T), or MicroSoft's WORD (select 'FORMATTED: NO' when saving), or WORDSTAR (non-document mode at the start) will do almost as well. EDLIN may be used and also COPY. A programmer might even choose to write such a file from BASIC. Most word processors will accept only a limited range of characters; control characters and graphics are usually not understood, or have some special effect. STEED can be used to enter such codes directly, which makes obtaining some features, like colour changes, particularly easy.

Notes on using PROMPT and ECHO are given in the MS-DOS book, although some help appears also in the Owners Handbook. ECHO may be followed by 'ON' or 'OFF' as shown, or by anything that is to be printed on the screen. Thus a special 'message of the day' could be incorporated or, more usually, a string of rather meaningless characters gleaned from the technical part of the handbook that sets up a colour scheme. Eg, the line

```
echo $[80;0;9;;~C
```

would set the 80 column mode with a light red background. The dollar sign in this sequence indicates the ESC character NOT the actual dollar character and cannot be obtained on some word processors. Confusingly, following the PROMPT command the dollar sign really does mean a dollar sign!

Finally, if a batch file is allotted the special name AUTOEXEC.BAT and is on the disk used to boot (start up) the system then it will be run automatically and can, of course, be used to configure the system suitably without bothering the user at all. On stand-alone systems with a number of different users it might be a good idea for each person to have his or her own disk which sets up the machine just as they need it.

# A Nimbus Case History

*by Keith Ashby*

In May 1985 we took delivery of a X10, the stand-alone winchester disk version of the Nimbus. The business includes agriculture, letting of cottages for self-catering holidays, letting apartments abroad and erection of holiday chalets. Previously an Apple had been used for the formulation of cattle rations. The temptation to put accounts on computer had been resisted, since we had a good manual system, and the only way to benefit from any saving in time would be to reduce the secretary's hours and loose the benefit of always having someone to answer the phone.

By the end of 1984, with the growth of the holiday side of the business and increasing complexity of the total business, the decision was made to computerise the accounts. Bearing in mind the Apple's age and in spite of it's reliability, a decision was made to purchase a new machine with facilities for accounts, database, spreadsheet, word-processing and communications over the PSTN via a modem.

Initially, consideration was given to the Mac and the offer of a trial machine for a few days was accepted. With only 128K available, it was unbearably slow. It appeared that it would be expensive to expand. In any case, I prefer a colour monitor!

The next logical alternative was an IBM compatible or more or less compatible which ran faster than a true IBM compatible. In the former category the Zenith 150 was offered at an attractive price, including software. In the latter category Apricot and Olivetti – both 8086 machines. The build quality on the Apricot did not impress nor did the attitude of Olivetti technical support staff, being very much on the lines of "ask your dealer" – and when you ask the dealer you find that he has to ask Olivetti!

Maybe these impressions are not representative of general experience, but it was at this stage we nearly placed an order for the Zenith, when the Nimbus appeared to rave reviews. In practice the competitive price that was mentioned in these reviews did not seem to be available to other than educational users. This made a machine of the configuration and with the software that we required barely competitive with the Zenith on price. However, in spite of the dealer being 70 miles away and against the advice of informed opinion (including that of the

Zenith dealer who said we would have to live with being on the cutting edge of technology!) we ordered a Nimbus X10 with 256K and mouse. The machine was supplied with Easy Junior Accounts package, Multiplan, Microsoft Word, Datamaster relational database and RM Basic all included in the package. The accounts program was the only one for which we received the proper manual, the others being copies!

In practice, all of the supplied programs were easy to install. MSDOS, with which I was not familiar, proved reasonably easy to learn. Mind you, late one evening I made the silly mistake of mixing up drives A and C in my mind and reformatted my winchester disk instead of the floppy, thereby destroying everything on the winchester disk! Fortunately, everything of importance was backed up, but the incident demonstrated the importance of using batch files for tasks such as formatting disks and running programs.

I now include the backup command in every major application batch file. Initially this backed up the whole program and seemed to take ages. Now, with the use of the /A and /M switch I only back up those files that have been opened since the last backup: it is much quicker. Every so often a complete backup is done using a batch file without the switches. This then sets free most of the backup disks that I've been using to store the record of files that have changed.

The value of this system was proved when on the Friday before I was due to go abroad for three weeks on the following Tuesday, the winchester disk developed a fault and would not run at full operating speed. A phone call to RML elicited the information that the repair could be done at 0830 on Monday. Whilst it was possible to send the Nimbus to RML by Securicor, it was improbable that it would be returned here on Monday in time for me to do the necessary work. In the worst case, depending on the fault discovered by RML's service department, this would entail me restoring all our software from backup.

The 130 mile drive from Kent to Oxford is made reasonably pleasant with the opening of the section of the M25 from the Reigate turnoff to the M40 and I arrived at RML by 0830. RML were, as usual, extremely pleasant and I walked out just after 1000 with a new winchester disk, all done under warranty. I started work again in the office at 1330 and by the end of the afternoon. with all the usual interruptions, I had restored all our software to the hard disk. Phew!

### Examples of Batch Files

Batch file to format a system disk.

```
REM THIS PROGRAM FORMATS A NEW DISK IN DRIVE A:
PAUSE INSERT DISK TO BE FORMATTED
FORMAT A: /S
CHKDSK A:
```

Batch file to backup MicroSoft Word.

```
REM BACKUP WORD DIRECTORY
PAUSE INSERT BACKUP DISK
CHKDSK A:
BACKUP C:\WORD A: /M /A /L
```

This backs up WORD from the directory called WORD on drive C (the winchester) to floppy on drive A.

/M means only backup those files that have changed since the last backup.

/A means add those files to be backed up to those already on the backup floppy and do not erase old files on the floppy.

The CHKDSK command ensures that the disk is still serviceable.

# Crossword Anagrams: for the Nimbus

*by W. Ashworth*

The position of some letters in crossword puzzle anagrams can usually be found by solving crossing words. Although it is not necessary to have these known letters, this program can take advantage if any are fixed by shuffling the remaining letters into the remaining spaces. It works equally well if the anagram result has to be in two or more words. The user is told how many arrangements are possible and is presented with a block of 15 or more (MOD X in line 837) at a time (in reverse video) another block being shown each time the space bar is pressed.

The especial interest of this compact program is that it incorporates a technique by which the program itself can write a line according to data input and then merge it ahead into the lsiting of the program as it runs (line 800), ready to be acted upon later. Obviously this possibility, which in Nimbus BASIC can be achieved by writing to a disk and immediately reading back again under the MERGEGO command (lines 2000 to 2060), is a valuable one for any application where the exact form of a line has to be calculated by the program itself. If a silicon disk is used it is almost impossible to notice any gap in the running of the program. A minor but useful technique in line 710 (PUT 8) suppresses unwanted screen output by overprinting letters.

```
  5  Count% := 1
 10  REM * ANAGRAMS - a program to help with crossword puzzles
 20  REM * by Wilfred Ashworth
 30  CLS : HOME
 40  PLOT "ANAGRAMS",10,180 SIZE 6,5
 50  SET CURPOS 20,12: PRINT "A PROGRAM TO HELP WITH CROSSWORD PUZZLES"
 60  SET CURPOS 50,18 : PRINT "by Wilfred Ashworth2
 70  REM * INSTRUCTIONS
 80  FOR I := 1 TO 2000 : NEXT
 90  CLS : HOME : PLOT "INSTRUCTIONS",5,220 SIZE 2,2
100  SET CURPOS 10,5 : PRINT "First please print the word in the form:"
110  PRINT:PRINT:PRINT"                              A*B*/C*D*" : PRINT
120  PRINT:PRINT:PRINT "   Where A,B,C,D....are known letters...in know
     spaces."
130  PRINT:PRINT "Asterisks represent unknown letters and slashes are used
     to divide into two or more words."
140  SET CURPOS 25, 16 : INPUT C$
150  REM *TRANSLATE THIS INTO A SUITABLE FORM FOR OUTPUT
160  DIM A$(15) : U$ := "" : M$ := "" : B$ := ""
170  DATA A$(1),A$(2),A$(3),A$(4),A$(5),A$(6),A$(7),A$(8),A$(9),A$(10),
     A$(11),A$(12),A$(13),A$(14),A$(15)
190  FOR T := 1 TO LEN(C$)
200    IFMID$(C$,T,1) <> "*" AND T <> LEN(C$) THEN U$ :=U$+CHR$(34) +
       MID$(C$,T,1) + CHR$(34) + CHR$(43)
```

```
210    IFMID$(C$,T,1) <> "*" AND T = LEN(C$) THEN U$ :=U$+CHR$(34) +
       MID$(C$,T,1) + CHR$(34)
220    IF MID$(C$,T,1) = "*" THEN READ M$
230    IFMID$(C$,T,1)  = "*" AND T <> LEN(C$) THEN U$ :=U$+M$+CHR$(34)
240    IFMID$(C$,T,1)  = "*" AND T  = LEN(C$) THEN U$ :=U$+M$
250  NEXT T
260  REM *ENTER THIS AS A LINE AHEAD so that letters can be substituted.
270  GOSUB 2000
280  PRINT:PRINT "Now type the remaining letters.....in the form UVXYZ"
290  PRINT:PRINT SET CURPOS 25, 23 : INPUT B$
410  REM * CALCULATE THE NUMBER OF ANAGRAMS
420  Q := 1 : L := LEN(B$)
430  FOR R := 1 TO L
440  Q := Q * R : NEXT R
450  PRINT: PRINT "THERE WILL BE  "; Q; "DIFFERENT ARRANGEMENTS"
455  PRINT
460  REM * WORD SCRAMBLING SECTION
470  DIM A(L), Z(L)
480  REM *PUT LETTERS FROM B $ IN ARRAY A$
490  FOR N := 1 TO L
500    A$(N) := MID$(B$,N,1)
510  NEXT N
520  J := L
530  GOSUB 560
540  PRINT : PRINT
550  PRINT : PRINT "      THESE ARE ALL THE POSSIBLE ANAGRAMS"
552  ERASE "ANA.BAS"
555  END
560  IF J = 1 THEN GOSUB 700: RETURN : REM *PRINT AND RETURN
570  Z(J) := 1: REM* INITIALISE AND START LOOP
580  J := J -1: REM* REDUCE LENGTH OF SUBSTRING
590  GOSUB 560: REM* GOSUB TO ITSELF
600  J := J+1:  REM* INCREASE LENGTH OF SUBSTRING
610  GOSUB 650: REM* ROTATE J CHARACTERS
620  Z(J) := Z(J) + 1 REM* INCREMENT LOOP
630  IF Z(J) < J + 1 GOTO 580:  REM* LOOP DONE?
640  RETURN
650  N := J: Ab$ := A$(N)
660  A$(N) := A$(N - 1)
670  N := N - 1
680  IF N = 1 THEN A$(N) := Ab$: RETURN
690  GOTO 660
700  FOR Y := 1 TO L
710    PRINT A$(Y); : PUT 8
720  NEXT Y
[800 U$ := A$(1) + A$(2) + A$(3) + A$(4)]
820  PUT 27, "[7m" : PRINT " "; U$; " "; : PUT 27, "[27m" :Count% :=
     Count% + 1
830  PRINT "  ";
835  ASK CURPOS V1,V2 : IF V1 > 65 THEN PUT 10, 13: PRINT
837  IF Count% MOD 15= 0 THEN PRINT: PRINT "Press Space Bar for more":
     PRINT :REPEAT : X := GET(): UNTIL X=32
840  RETURN
2000 CREATE #12, "ANA"
2010 PRINT #12, "800 U$="; U$
2020 CLOSE #12
2030 OPEN #12, "ANA"
2040 MERGO "ANA", 280
2050 CLOSE #12
2060 RETURN
```

This technique whereby a program can write one of its own lines while running and merge it ahead in time to be used can be done directly on the NEWBRAIN computer but in RML BASIC I have only been able to implement it by writing to disk and immediately reading back again as a merged line. If anybody knows how it could be done directly I would love to know how!

Price List July 1986

# Whittle Systems

## Turbo Charge your Nimbus

| | Education | Commercial |
|---|---|---|
| **SI – TURBO** | | |
| – 511k Silicon Disk drive | £25 | £25 |
| **WS 20XW** | | |
| – 20 Megabyte external Nimbus Winchester | £925 | £1200 |
| **WS20 FW** | | |
| – 20 Megabyte external Nimbus Winchester with 5.25 inch floppy disk drive | £1115 | £1395 |
| **WS40 XW** | | |
| – 40 Megabyte external Nimbus Winchester | £1695 | £2225 |
| **NVD – DISPLAY** | | |
| – Nimbus VIEWDATA display & terminal emulator | £49.95 | £75 |
| **TRAPDOOR** | | |
| – Trilog Software's front panel for Nimbus | £49.95 | £49.95 |
| **CP/M 80 Programmer's Guide** by Morrell & Whittle | £8.95 | £8.95 |

Network prices of software up to

| | |
|---|---|
| Up to 8 stations | @ 3 × standalone price |
| Up to 16 stations | @ 4 × standalone price |
| Up to 24 stations | @ 5 × standalone price |
| Up to 32 stations | @ 6 × standalone price |

*All prices exclude P & P and Vat.*

**9, Herschel Crescent, Littlemore, Oxford OX4 3TS** Tel 0865 711956

# Communications – or going on-line

*by Robin Davies*

The DTI initiative in buying up large quantities of surplus modems to issue to schools will have one, and only one, major consequence for those members who own BT shares. For those who only have BT bills there will be a rather different consequence. More importantly, it will push into focus the whole question of on-line communications – to use some more jargon. In *Read Using 14* I expressed a view that such computer-to-computer communications were important. Since then I have been exploring the idea from both a theoretical and practical viewpoint.

If your school receives a modem and gets it operating, what can be done with it and how do you go about this business of on-line communications?

First of all, what do you need? Well, apart from the modem, you will need a telephone line – preferably dedicated and not through the usual antique school switchboard. You will also need a computer and most importantly the right software.

Let's deal with the modem first. Now they've been around for years and the word itself is a combination of *modulater* and *demodulater*. In 1980 the cheapest modem available in the UK was around £1500. Today you can buy one for less than £50. Without going into great technical detail, the job of a modem is to convert what comes out of the computer into signals that can be sent over the telephone lines and understand by another computer at the other end of the line. At the moment, most modems either use V21 or V21 & V23 CCITT standards. That more or less means that a modem transmits and receives information at 300 baud, 1200 baud and 1200/75 baud. I do mean *baud* and not *bored*. The word refers to bits of information transmitted per second and comes from the name of J.M.E. Baudot, a French telegraphic engineer who worked out how to send messages along a wire in the middle of the 19th century.

To help make life simple, there are two sorts of modem: acoustic and hard-wired. The acoustic modem is portable and has two rubber cups, one at each end, into which you push your telephone handset. The hard-wired modem – which is the sort supplied under the DTI scheme – makes a permanent connection between the telephone system and the computer. The latter type of modem is more reliable as it is less prone to problems caused by noise and poor connections which are sometimes a feature of the acoustic modem. I have been using an acoustic modem for a while now with no problems. Mind you, it was one reduced from £250 to £40 so the quality is good.

Of course, you must connect the telephone line to the modem and the computer to the modem. The telephone and the computer plug into the modem and the modem plugs into the telephone wall socket. The computer and the modem should be connected to each other by means of a standard RS232C female socket on each device. Naturally, RML don't use a standard RS232C device on the 480Z so you will need a suitable cable – bought or made. It costs about £5 to make one and perhaps 4 times that to buy one, so get out your soldering iron. It's very easy to put a cable together as although the RS232C interface has 25 pins, you actually only use very few pins for communication.

The DTI modems will enable you to use such exciting things as PRESTEL and MICRONET 800. These use a split baud rate of 1200/75. This means that you receive information at 1200 baud and transmit at 75 baud. The designers of PRESTEL never imagined that users would wish to send lots of information but would be receivers primarily. This is, I feel, a major drawback. The latest RML Viewdata package contains the necessary software to allow you to use the DTI modems for contacting PRESTEL.

We come now to consider why PRESTEL is rather a 'noddy' system and therefore the need to consider a far wider field than PRESTEL will ever offer. Here we must consider why there is a need to go on-line at all. Are there perhaps two reasons? First, to teach children the skills of on-line retrieval in an age where such a skill will be of increasing importance and second, for a school or college to use the facilities available world-wide for the retrieval of information. If the concept of going on-line world-wide seems daunting, fear not. It isn't, as we shall see later.

For the purposes of developing on-line skills or acquiring serious information, PRESTEL has limitations. For older students, such as those studying for A-level and beyond, PRESTEL is of little value. Such needs have to be fulfilled elsewhere.

At this point the on-line retriever turns to BLAISE, DIALOG or THE KNOWLEDGE INDEX. These are large electronic databases accessible via the telephone for the cost of a local telephone call (and a little more, one must confess!).

BLAISE is the *British Library Automated Information Service*. Many of the major public and university libraries have on-line access to BLAISE but it is a facility available to anybody and offers two separate facilities: BLAISE-LINE and BLAISE-LINK. LINK is primarily a collection of medical and related databases, while BLAISE-LINE is the service which provides bibliographical information about publications. Thus if you or a student needs information for a project on aircraft markings, BLAISE can give details ef *every* book and magazine which has or had information about that or any other subject.

DIALOG is the world's largest storehouse of electronic information with more than *200 separate databases*, 90 million items of information and 100 gigabytes of total storage capacity. Yet it can take only 8 or ten seconds for the DIALOG mainframes to sift through all of that information and retrieve thousands of 'hits' on a particular word.

THE KNOWLEDGE INDEX was launched in Britain in 1984 and is essentially a truncated version of DIALOG offering a quarter of the databases. It was set up specifically to offer a low-cost way into DIALOG, which can be expensive to use with a ten-minute search on an expensive databases costing upto $25 dollars, including communications charges.

So, now we come to one of the most important factors in considering the feasibility of going on-line: the cost. In education, where pennies matter, on-line costs need to be kept down.

The databases mentioned above including two which are in California, BLAISE is in the south of England. Thus, even if you are ringing BLAISE from Newcastle, you are apparently involved in long-distance telephone calls before any costs from the database itself. Well, not quite. In fact by using British Telecom's data network, Packet Switch Stream (PSS), you search any database for the cost of a call to your local PSS node. This means that from Newcastle you dial 0632 314171 or 314181 and then enter a string of numbers and letters which are your Network User Identity, your Network User Address followed by your database password. To use BLAISE you must be a registered PSS user.

To join PSS needs an initial subscription of £25 plus a quarterly rental of £6.25. In addition, there are connect and volume charges. All this means is that PSS would charge you about £2 to connect to an American host for a ten-minute search (including VAT).

If you join DIALOG or THE KNOWLEDGE INDEX there is no need to join PSS as the DIALOG company has negotiated with PSS to include PSS charges within DIALOG charges. It cost nothing to join DIALOG and in fact, new subscribers are given $100 worth of free time to explore the system. What you do pay for is the cost of searching the individual DILAOG databases on an hourly basis. Charges range from $35 to $180 per hour, plus charges levied by DIALOG. Joining THE KNOWLEDGE INDEX costs £25 initially and gives you 2 hours free connect time with a flat-rate connect-hour charge of $24 after that.

If you join PSS then you can, for the cost of a local telephone call to you PSS node, search two fascinating databases in America call THE SOURCE and COMPUSERVE. These are general interest, hobbyist databases which make MICRONET 800 look very small.

Nearer home, there are dozens of private Bulletin Boards (BBSs) around the UK which offer on-line facilities free of charge, except for your telephone charges. Details of these are published regularly in *Practical Computing* and *Personal Computer World*. Many are available 24 hours a day and are fascinating to access. You don't need and can't use PSS for these.

All these mentioned above are accessed at 300/300 baud and need software which will store information to memory and disk whilst they are being used. Such communications sofware is freely available. The two best programs are KERMIT and UKM7. Contact me for details. KERMIT is available configured for the 480Z and Nimbus. UKM7 can be configured for most micros.

Finally, and perhaps most interestingly is a recent development, MICROLINK. This is a communications facility available via Telecom Gold and PSS and is developing rapidly. It seems to be very cheap, costing £5 to enrol and £3 per month plus connect charges. A section of MICROLINK is being set aside for RMNUG users (called a Closed User Group). I encourage you to write for free details from:

Database Publications Ltd.,
Europa House,
68, Chester Road,
Hazel Grove,
Stockport, SK7 5NY

NVD – DISPLAY V 2.00A

# Whittle Systems

## Nimbus Viewdata

Editor available shortly

## Display

- Full 40 Column Prestel terminal emulation
- 40 column scrolling terminal emulation
- 80 column ADM3 scrolling terminal emulation
- Autodials via – DTI Modem
  – v25 bis Modem
- Send & receive text files
- Save frames to Jotter file
- CET Telesoftware downloaded
- CET FIF format viewdata databases
- Free Modem cable to connect to DTI modem
- Carousel for free standing demonstration

Single Station licence

£49.95 education
£75 commercial

## Editor

- Simple to use
- Very powerful
- Create & modify your own databases
- Incorporate frames from Jotter with your database
- Over 500 frame database on PC1 or over 16,000 frame database on WS2OYW
- Compatible with Micro Viewdata
- Menu selection of viewdata attributes
- Cut, copy, paste window
- Show all control codes
- 3 levels of undo
- Cross wire or text cursor mode
- Grid to help with layout
- Extended cursor mode for column entry
- Word wrap

Single Station licence

£95 education
£175 commercial

(Editor price includes DISPLAY licence)

Network prices of Software

Up to 8 stations @ 3 × standalone price
Up to 16 stations @ 4 × standalone price
Up to 24 stations @ 5 × standalone price
Up to 32 stations @ 6 × standalone price

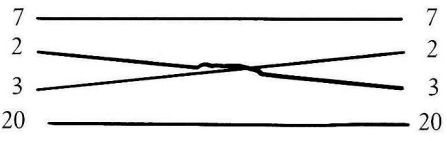
*All prices exclude P & P and Vat.*

**9, Herschel Crescent, Littlemore, Oxford OX4 3TS** Tel 0865 711956

# A Simple Program to access your local bulletin boards

Do you have an old modem or acoustic coupler connect it to the 380Z and access some of your local bulletin boards (300 baud only ) all for the cost of a telephone call. ists of the telephone numbers can be found in most popular computer magazines.

Here are the pin connections for the cable to connect the acoustic coupler or modem to the 380Z .




Computer
380Z
SIO4

Acoustic coupler
or
Modem

```
                ;A VERY SIMPLE COMMUNICATIONS PROGRAM

                ;J.H.MASON
                ;WESTMINSTER CITY SCHOOL SW1

                ;EMT'S
0001 =          OUTC    EQU     01H     ;output to screen
0005 =          LPOUT   EQU     05H     ;output to port
0029 =          SETLST  EQU     29H     ;set printer options
0030 =          S4KIN   EQU     30H     ;get character
002F =          S4KTL   EQU     2FH     ;check sio4
0021 =          KBDW    EQU     21H     ;wait for character
001F =          KBDTL   EQU     1FH     ;check keyboard
0004 =          PORT    EQU     04H     ;SIO4 port
0003 =          RATE    EQU     01H     ;baud rate set at 300
0017 =          MSG     EQU     17H     ;message to screen

0100                    ORG     0100H   ;load program at 100H

                ;****************** SET PORT AND RATE *******************
0100 3E04               LD      A,PORT  ;select SIO4
0102 1E01               LD      E,RATE  ;select baud rate
0104 F729               EMT     SETLST

                ;********************* CLEAR SCREEN **********************
```

```
0106 210B01   SCLEAR: LD      HL,MESS ;start of screen
0109 F717             EMT     MSG     ;clear screen
010B 1F0D     MESS:   31,13

010D 2042554C         ' BULLETIN  BOARD  PROGRAM                V1.1    JHM ',13
0143 203D3D3D         ' ========================',13,13,13
                ;*************SCAN KEYBOARD AND SERIAL PORT **************
0160 F71F     KEY:    EMT     KBDTL           ;Test keyboard for character
0162 CA6B01           JP      Z,SIO           ;Loop if no character
0165 F721             EMT     KBDW            ;Get character from keyboard
* 0167 F701           EMT     OUTC            ;Remove this for full duplex
                                              ;otherwise 2 of every
                                              ;character will appear on the
                                              ;screen
0169 F705             EMT     LPOUT           ;Send character to serial port

016B F72F     SIO:    EMT     S4KTL           ;Test sio4 for character
016D CA6001           JP      Z,KEY           ;Loop if no character
0170 F730             EMT     S4KIN           ;Get Character from sio4
0172 F701             EMT     OUTC            ;Print to screen
0174 C36001           JP      KEY
                ;*******************************************************


000F CLEAR   001F KBDTL   0021 KBDW    0160 KEY     0005 LPOUT
010B MESS    0017 MSG     0001 OUTC    0004 PORT    0003 RATE
0030 S4KIN   002F S4KTL   0106 SCLEAR  0029 SETLST  016B SIO
```

No errors

* This line is used when the board is running half duplex otherwise the characters that you type will not appear on the screen.

Connnect the Acoustic coupler to the SIO4 port on the 380Z with the cable connections shown above, load the code into the computer buy either using the front panel or use a text editor and zasm to assemble the code and create a COM file.

On execution of the program it will display a caption initially and the appear to do nothing,when in fact its continually reading the keyboard and the SIO4 port for incoming data. Dial the number of the bulletin board you want to access and place the headset in the acoustic coupler, wait for the coupler to lock on to the carrier, the data should appear on the screen in a few seconds.

This program will allow you to access most of the bulletin boards, with the exception of PPS operating at 300 baud provided you have a good strong signal and your equipment is functioning correctly . You cannot download any data with this program it can only be used to access.

# Micro Market

This is one of the many services offered by the User Group. If you have any RML equipment that you want to buy or sell, then why not try the Micro Market? RML's sound hardware will probably last for ever and should not suffer from being secondhand.

If you have secondhand equipment that you want to sell please write to the Secretary who will ensure that it is included in the list.

This list is now going to be prepared fresh for each issue of *Read Using* and there will be no items carried over from one issue to the next. This is because previous requests to notify the Secretary have been ignored with the result that some prospective purchasers have telephoned after equipment long since sold!

**Please included your membership number with each Micro Market entry**

List of equipment for sale or wanted – July 1986

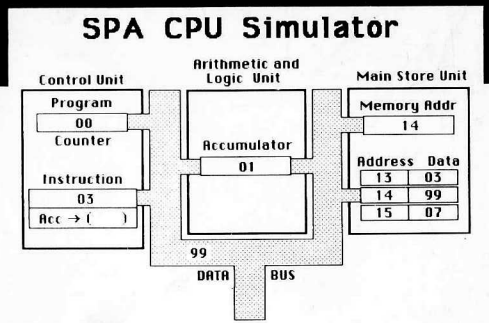
Wanted – *cheap* 380Z. Any state of repair, but must have a good case and power supply. Ring **Lyndon** (0443) 690174.

380Z, 56K, FDS-2, 9" b/w monitor, HRG and 40/80 boards, SIO-4 and centronics ports, TXED, FORTRAN, WORDSTAR, LINKUP with disks and manuals. Offers to Martin Range at Computing Unit, The Social Studies Faculty Centre, George Street, Oxford. OX1 2RL. (0865) 724747

380Z Network Server with twin 8" disks. Contact Mr. Williams at St. Simon Stock School, Maidstone, Kent. Tel: Maidstone 54551

★ ★ ★ **Remember** ★ ★ ★

Deadline for next issue is September 12th




# CPU-SIM

**3/480Z £15**
**480Z Network £39**
(380Z needs HRG)

A brilliant demonstration of how the heart of a computer works. CPUSIM is an animated colour graphic simulation of a Central Processing Unit. It shows in detail how each part of the 'Fetch-Execute' cycle works. You can load or enter programs written in a simplified machine code language. Then watch how the CPU fetches instruction codes, interprets them, fetches additional data if necessary and performs calculations or other operations. You can inspect and set memory locations (a bit like the 'front panel')
If (like our own programmers) your students are faintly worried by 'addressing modes', a spell with CPUSIM is just what they need!
Naturally CPUSIM is very simple to use. Proud owners of the ILECC buffer box can take full advantage of flashing lights, pressing buttons, etc.

## Extensions to Nimbus RML software

*We have a number of extensions available for RM BASIC and RM Logo. These include:*

**Graphics Toolkit** (for Nimbus BASIC)
**Nimbus £15 Nimbus Network £35**
Provides picture loading and saving, user definable character sets, user definable markers, and commands to copy areas of the screen to and from memory.

**Logo Gallery** **Nimbus £13 Nimbus Network £29**
An extension to RM Logo that lets you keep pictures on disk and fetch them again later. As well as fetching Logo pictures, you can fetch pictures created by PaintSpa or Basic as well – you can also load your Logo pictures into PaintSpa!

**Execute** **Nimbus £15 Nimbus Network £35**
*Execute any MS-DOS command from within Basic!*
Enables you to perform MS-DOS command lines from BASIC!
For example, from within Basic you could run PaintSpa to paint a picture; load a full screen editor and then edit the Basic program itself; Use PRINT to list programs in the background while carrying on running your own program; load Multiplan, Logo, Word etc.
When you've finished, you can go straight back to BASIC!

*Please add VAT at 15% to all prices.*

SPA

**P.O. Box 59 Leamington Spa CV31 3QA Tel: 0926 22959**